Deliverable

D3.1

Models and Framework for Metadata Generation and Policy Infrastructure

WP3 Meta-data generation and policy infrastructure

December 2008
Version 1.0

Consequence

Context-aware data-centric information sharing

FP7-ICT-2007-1
ICT-2007.1.4. Secure, dependable and trusted Infrastructures
Grant Agreement 214859

LEGAL NOTICE

The following organizations are members of the Consequence Consortium:

Europäisches Microsoft Innovations Center GmbH,

BAE SYSTEMS (Operations) Limited,

Hewlett-Packard Italiana,

Imperial College of Science, Technology and Medicine,

The Science and Technology Facilities Council,

Consiglio Nazionale delle Ricerche,

Centre for Research and Telecommunication for Networked Communities.

Project acronym: Consequence

Project full title: *Context-aware data-centric information sharing*

Work Package: **3**

Document title: **Models and framework for Meta-data generation and policy infrastructure**

Version: **3.1**

Official delivery date: **31 December 2008**

Actual publication date: 19 December 2008

Type of document: Report

Nature: Public

Authors: Vaibhav Gowadia, Enrico Scalavino, Emil Lupu (IC), Benjamin Aziz (STFC)

Approved by: Dmitry Starostin, Alexey Orlov, Joris Claessens (EMIC), Ilaria Matteucci (CN)

Version	Date	Sections Affected
0.1	8 October 2008	Initial draft on policy infrastructure.
0.2	14 October 2008	Added Section 3.1.2 (Benjamin Aziz)
0.3	27 October 2008	Added Section 2.1.2 (Benjamin Aziz)
0.4	07 November 2008	Added Sections 3.2.2, 4.2 and 4.3.2 (Benjamin Aziz)
0.5	19 November 2008	Revised Sections 2.1.1, 3.1.1 and 4.1. Draft released for internal review.
0.9	15 December 2008	All sections. Almost final draft.
1.0	23 December 2008	Updated Sections 2.1.2 and 4.2.6.

Table of contents

1 Introduction

Enforcement of data sharing agreements requires that the security requirements of the agreement are expressed as enforceable policies, and these policies can be evaluated in a cross-domain scenario. This document focuses on policy infrastructure and metadata generation infrastructure to support these requirements.

Organizations depend on data sharing for varying reasons. While they need to share data, they want to control how the shared data is used after it has been given to another organization. The requirements to control usage of data may originate from several requirements, like the need to protect sensitive data, protect commercial interests of the organization; provide accountability, or the need for timely dissemination of data. The problem of sharing data in cross-domain scenario presents a combination of several problems, like access control, rights management, trust management, privacy, etc. A key requirement for controlling the usage of data is to provide *continuity* of control. This problem is called *usage control*. In past, research has focussed mainly on individual problems mentioned above. There is a lack of architecture that allows flexibility to addresses these problems together. We have given an overview of architecture to address these needs in deliverable document D1.1. This document describes policy infrastructure that is needed to support the Consequence architecture. We discuss an enforceable policy language capable of expressing the requirements seen in the testbed scenarios. Note that in this document we always talk about *enforceable* policies even when we may not say so explicitly each time. We also identify components of the policy infrastructure and discuss how they should interact with other components of the Consequence architecture.

Meta-data is an important ingredient of policy infrastructure. It can be used to describe the characteristics of data, like who owns the data, how the data was collected, what the data is about, etc. It can also be used to describe an organizations structure, users, and their attributes. In the policy infrastructure, metadata plays the role of glue that binds policies with the protected data and its users. To enable enforcement of policies across organizational boundaries, the organizations sharing data should agree on a common metadata vocabulary and associate metadata with protected data. Therefore, in this document we also discuss the requirements for metadata generation and propose a methodology to be used in Consequence.

The rest of the document is organized as follows. In Section 2, we first discuss the requirements for policy infrastructure and metadata generation to support the goals of data sharing and usage control. We also identify requirements that should have higher priority in development of policy infrastructure. In Section 3, we present related research work and discuss their shortcomings. Finally, in Section 4, we propose a policy language, components of the policy infrastructure, and infrastructure for metadata generation. We also analyze features of the proposed models against the requirements identified in Section 2.

2 Subsystem infrastructure requirements

2.1 Categories

2.1.1 Policy Infrastructure Requirements

The Consequence project focuses on data sharing and usage control requirements in the context of protecting sensitive data and in crisis management scenarios. While work packages 5 and 6 focus on defining test bed scenarios and requirements for crisis management and sharing sensitive data respectively, in this section we discuss requirements from the perspective of policy specification, policy evaluation, and implementation of policy decision

points. In addition to the testbed scenarios, we have drawn additional requirements from many real data security policies and agreements (BBSRC; CCLRC, 2002; DCA, 2003; Medical Research Council, 2003) and the test bed scenarios, and have identified the requirements for policy infrastructure. The important requirements include cross-domain interoperability, ability to express contextual conditions over authorizations, obligations based on user access and other events, ability to evaluate policy when the recipient has no longer network connectivity with the sender, and ability to specify trusted credential issuers. A more detailed discussion of the requirement categories identified is given below.

2.1.1.1 Cross-domain Interoperability

Organizations sharing data may act as a sender or receiver. The organization where a document originates is called the *data provider* and the recipient organization is called the *data consumer* of that document. The access restrictions for the protected document originate from the data provider but they have to be enforced in the data consumer's domain. Therefore, it is very important that the interpretation of protection requirements in the sender and recipient organizations be consistent. Achieving cross-domain interoperability requires the use of attribute-based rules and trust relationships.

Attribute-based Rules

To resolve the interoperability problem data sharing agreements often describe protected content and subjects by their attributes. Attributes based rules are required to describe either authorization rules or obligation rules. For this strategy to work, at the policy level[1] it is implicit that the organizations participating in data sharing establish a common vocabulary. Following is an example from the Sensitive scientific data test bed that specifies an attribute-based authorization rule.

Example 1. "*An ICAT Administrator ... can also update, logical and physical delete DataEHs that are new or it owns, as long as these Data EHs are not set as 'Facility Acquired'. This attribute denotes data and metadata imported from other systems.*"

Following is another example from (Medical Research Council, 2003) specifying an attribute based authorization rule.

Example 2. *"Researchers must ensure that personal information is handled only by health professionals..."*

In above example, subjects are described by an attribute about their profession and the protected content can be described by an attribute that specifies whether that data item is considered personal information or not. To be able to express such policy rules, it is required that the policy language caters for specification of both subject attributes and object attributes.

Trust Relationships

Evaluation of data sharing policies requires that organizations sharing data establish certain trust relationships with each other. The purpose or function for which an entity is trusted may be different for each entity involved. From the perspective of policy infrastructure requirements, two major categories of trust relationships can be distinguished: 1) trust for evaluation of policy, and 2) trust for verifying user credentials. A data provider may need to trust the recipient or another external entity to correctly evaluate its policy, and a policy evaluator may need to trust external security token services for correctly determining user's attributes or verifying user's credentials.

[1] Note that although a common vocabulary is necessary, the implementation of the policies may differ.

While the data provider specifies the policy for making authorization decision, the policy may need to be evaluated in its own organization or in the data consumer organization. Suppose a user of data consumer organization makes an access request. If the data provider evaluates the policy, then the data provider has to trust the data consumer for correctly verifying user credentials. Otherwise, if the data consumer evaluates the usage control policy, then the data provider has to trust that the data consumer is doing it faithfully and correctly.

Let us consider an example scenario where the user requesting access to protected content does not belong to the organization evaluating the security policy. The Crisis Management test bed such scenario in following statement.

Example 3. "*The act also acknowledges that other types of organizations may be called upon to help in an emergency. It should also be noted that some of these responders dealing with the incident may be public agencies and some may be commercial organisations".*

For instance, in a crisis management scenario, health professionals from external hospitals may be brought in to help at a hospital. In such case, evaluation of security policy will have to trust external security token services for correctly determining user's (external doctors) attributes or verifying user credentials. Thus, the authorization mechanism needs to be flexible in the sense that it should be able to accept assertions about subject identity and attributes from external services.

Following is an example from the Sensitive Scientific Data test bed, which illustrates need for policy and enforcement infrastructures to be able to trust credentials from more than one source. In following case, the policy infrastructure must be able to trust attributes provided for users visiting from other research units.

Example 4. "*We envisage that researchers from each organisation work in the same research unit and share IT facilities.*"

From the above discussion of required trust relationships, we can infer the following two requirements for the policy infrastructure: 1) an enforceable usage control policy must be able to specify which certification authorities are trusted to certify values of which attributes, and 2) the policy infrastructure should be able to verify whether acceptable authorities issued the attribute certificates provided with a usage request.

2.1.1.2 Authorizations

Expressing authorizations rules or access control constraints is one of the main goals of the data sharing policies. The requirements of expressing access control rules in cross-domain scenario include the requirements of traditional access control in single domain. Well-known requirements for expressing authorizations rules include ability to express permissions and denials, ability to specify rules applicable to all subjects, ability to specify propagation of permissions, and ability to enforce conflict resolution rules when conflicting authorizations can be derived.

However, in data sharing scenarios the authorization rules should allow more flexibility in expressing authorization rules and the policy infrastructure should also perform dissemination control in addition to controlling access to data. We now discuss these requirements in more details.

First, more flexibility is needed from the perspective of data provider because the data provider does not directly know or trust all the users in data consumer (recipient) organization. Thus, the policy language should be able to specify trust constraints that describe how to trust a user in the recipient organization. From the discussion in Section 2.1.1.1, we already know that the data provider should be able to define constraints specifying

which users can vouch for identity and attributes of a subject. In addition to these trust constraints, the specification of authorization rules should be able to specify additional trust requirements like *third-party approvals, threshold based authorizations,* or possession of multiple security tokens of other kinds.

- **Third Party Approval**: Besides the explicit data owner's consent, third party explicit approval is often considered a requirement for data usage. The third party whose approval is needed, may be a supervisor/guarantor entity (e.g. for privacy law enforcement), a work partner who shares rights on the data, an entity holding the rights on the data, or an entity being delegated to take decisions on behalf of the actual holder (e.g. a relative for an unconscious or untraceable person, an attorney etc.).
- **Threshold based authorizations**: In many situations, a data provider may want to require the user to produce multiple security tokens (e.g., proof of approval) from a minimum number of distinct authorities to allow access to the user. Examples of such situations include a user accessing very sensitive data, and a user accessing data whose protection is of interest to multiple entities.

Following is an example from the Sensitive Scientific Data test bed, which requires multiple approvals for using personal information in scientific experiments.

Example 5. "*the use of identifiable patient data is subject to the Health and Social Care Act 2001 (England and Wales) plus approval from the appropriate research ethics committees and governing bodies like the Caldicott Guardian.*"

Second, there is a need for controlling dissemination of data when data sharing is done. Many organizations implement custom software to completely automate the process of dissemination. However, such software solutions are hard to maintain, as they need to be modified when policies change. While the Consequence architecture supports automated dissemination of data based on events, it is not reasonable to expect all dissemination to be automated. It should be possible to selectively disseminate data based on request of data consumer or a user is the data provider organization. Therefore, it is crucial that the policy infrastructure is able to check whether data dissemination is permitted by the data sharing agreement.

In addition to the requirements above, authorization rules applicability may be restricted based on conditions. There are requirements for many types of conditions that can be specified in an authorization rule. Also, conditions may need to be specified with some obligation rules. We defer discussion of conditions to Section 2.1.1.4.

2.1.1.3 Obligations

Obligations are actions that must be performed by an entity. An entity (user or an organization) may be obliged to perform obligation actions when certain events occur, including an access request. The entity may be required to perform an obligation action before, during, or after an access request. In usage control terminology, these obligations are called pre-obligations, ongoing-obligations, and post-obligations respectively. Pre-obligations are actions that are required to be performed after an authorization has been granted but before the requested access is actually allowed. An example of a pre-obligation is to log usage request before usage is allowed. Ongoing-obligations are actions that are performed during the duration of usage request. For example, deduct one point from user account after every one-minute during a call made using a calling card. A post-obligation is performed after the usage request is complete. For example, delete the document after reading it. Data sharing agreements often specify obligation actions that are associated with access authorizations.

Example 6. *"to log access to certain types of document that include sensitive personal information".*

Above example is from the crisis management test bed. It specifies obligation actions that are dependent on accessing a document. Thus, the policy language must be able to express obligations actions along with any parameters needed to specify the obligation action.

A more general requirement to express obligations is based on events and satisfaction of certain conditions. Obligations may be described based on temporal events, non-temporal events, or a combination of both type of events. Temporal events specify a specific time and date. For example, on first day of each month a bank should share the total amount of transactions made by each account holder in the previous month with the credit rating agency. Non-temporal events are based on change of a system's state, e.g., learning new facts or modification of existing facts is considered an event. Typical example of such non-temporal event in data sharing agreement is the requirement for the recipient to perform obligation action when it receives data from the data provider. In other words, the recipient is expected to act on information it receives. Following is an example from crisis management test bed, which specifies an obligation to delete certain documents when system state changes from "emergency" to a non-emergency state.

Example 7. *"delete the documents after the emergency is finished".*

The data providers are typically obliged to share new information or provide updates to previously shared data. Following is an example from (BBSRC) illustrating such requirement.

Example 8. *"all data should be shared in a timely fashion as soon as it is verified."*

Events that trigger execution of an obligation can also be combination of temporal and non-temporal conditions. For example, consider the following rule: the recipient should delete the shared document 30 days after receiving it. The requirement applies even if no access request is made.

2.1.1.4 Conditions

Data sharing agreements may specify several types of conditions over authorizations and obligations expressed in it. The enforceable policy must be able to express these conditions and the policy infrastructure must be able to evaluate them. The design of policy infrastructure depends on where, how, and when these conditions can be evaluated. Therefore, based on these criteria we categorize the various types of conditions as contextual conditions, monitored conditions, simple conditions, history-based conditions, and security token-based conditions.

Contextual Conditions

By context we mean circumstances under which the usage request is made. Context of request is established when a request is made and is reliably known only at the point where the request is intercepted for authorization evaluation. Some of the contextual properties (e.g., role of a requestor) may be received with the access request itself, while others may have to be obtained from environment, or other services. Examples of contextual properties include user location and time. An example of an authorization rule using a temporal constraint is, do not allow external users to access project data for three years after end of project. An example of obligation rule including temporal conditions is, "delete data 30 days after receiving it".

Contextual properties can also include other circumstances as amount of available resources at the time of request or status of access device. Contextual conditions are often specified in

data security policies with the goal of reducing risk of unauthorized access or misuse of data. For example, the following policy from (DCA, 2003) shows an example of context-dependent permissions:

Example 9. *"Personal information disclosed must: ... e) be located in a geographically secure environment, f) not be inputted/accessed without industry standard security devices".*

Following is an example of location-based condition used in an authorization rule in the Sensitive Scientific data test bed:

Example 10. "*All work related to the development of its drug discovery software in Phase 2 of the project must be: carried out in its laboratory ...*".

Monitored Conditions

Usage control (Park & Sandhu, 2004) is often context sensitive in ways that may require monitoring of system parameters or contextual parameters. For example, a security policy may specify to allow access only *while* the system has at least 256 MB of free memory, or *while* the user is in office. To evaluate and enforce such usage control policy, it is necessary to actively monitor the free system memory and user location. Furthermore, an ongoing access should be terminated if the specified monitored condition becomes false. Following is an example of requirement from the Scientific Sensitive Data test bed that seeks to monitor contextual parameters.

Example 11. "*The physical usage contexts (e.g. time, geographical location) would evolve dynamically over the workflow enactment, the proposed solution must therefore be capable of monitoring environmental parameters to ensure the correct enforcement of context-related low-level data policies at read time".*

Simple Conditions

Simple conditions are conditions that need to be evaluated only before granting the access. An ongoing access request need not be interrupted if a simple condition becomes false after access was granted. Policies often specify simple conditions over subject and object attributes. A sub-category of requirements where simple conditions are specified is *explicit consent.*

Explicit consent is one of the most important requirements for personal data usage. Most national legislation about privacy or medical information management policies considers explicit consent as the basic condition for data usage. The following example is taken from (Medical Research Council, 2003)

Example 12. *"Research should therefore be designed to allow scope for consent, and normally researchers must ensure they have each person's explicit consent to obtain, hold and use personal information."*

In particular it is possible to distinguish between the explicit consent of the data owner (i.e. who has commercial rights share over the data) and the explicit consent of the data subject (i.e. who the data refers to). If MRC produces a set of testing data referring to some patients' treatment, the data owner is to be considered MRC itself, while patients are the data subjects. The consent values can be stored in a database and can be referred by using an attribute in the policy. Consent values do not change frequently and from the perspective of performance of policy infrastructure it is a good choice to express constraints over consent as a simple condition.

History-based Conditions

Another factor that some security policies consider is data usage history. Many usage policies put a numerical or temporal limit to the access. A user's usage and behaviour history could also be used to calculate the user's trustworthiness or the risk associated to a specific information disclosure. In some research scenarios described in (Boddy, 2004), data sets can become so large that it is necessary to eliminate rarely used data in order to reduce risk:

Example 13. *"Resources should be concentrated on the more heavily-used data sets. Past usage is an appropriate guide to what should be provided and levels of investment in different data sets."*

Purpose Awareness

High-level usage control policies often specify a purpose for which access is allowed. However, an implementation of a usage control system may not be aware of the purpose for which the access is requested. In such cases, requirements for purpose should be refined into attribute based conditions that can be evaluated by the system. One way of ensuring that data is used only for specific purpose is through design of special roles. The policy writer must create an access control role that provides permissions to perform operations allowed under a given purpose. Access to protected data must be restricted to such special roles. Only applications that perform permitted operations should be able to acquire these roles. If users acquire these roles or have direct access to data, the purpose for which a user may use data after gaining access is beyond the scope of policy infrastructure. Note that solution to this problem relies on design of policy rules and roles, automating data processing for specific purposes, or trusting end users. Purpose-based conditions for access specified in high-level policies do not entail any new requirements for policy language and policy evaluation.

2.1.1.5 Policy Evaluation

Deterministic Evaluation

Any organization that participates in data sharing will have its own organizational policy, as well as policies derived from data sharing agreements it has. This raises the concern of resolving conflicts among these policies. The policy evaluation process should always return deterministically derived decision or an error. A usage control system should be able to detect and resolve such conflicts or report them to data consumer.

Partially Offline Evaluation

Requirements from the test bed scenarios indicate that it should be possible to evaluate policies when the sender organization is no longer reachable from the recipient organization. This requirement for policy infrastructure originates from two needs of organizations. First, organizations do not want their activities to completely depend on availability of services in other organizations. The goal is to minimize cross-domain dependencies for policy evaluation and enforcement after data has been exchanged. Second, existence of communication channels with other organizations create opportunities to exploit these channels as covert channels for leaking sensitive information. While this requirement is good from above perspectives, it conflicts with other policy requirements. For example, to be able to revoke a policy the policy enforcement layer in the recipient organization must be able to lookup/receive the revocation list or an updated policy from the sender organization. Since, these requirements conflict only one of them can be fulfilled at any given time. The policy infrastructure can allow the flexibility to accommodate either of the requirements, but the policy writer must specify whether it is required to check for revocation or not.

2.1.1.6 Threat Mitigation

While protection of sensitive data is responsibility of the enforcement layer, the policy evaluation process must be designed to mitigate certain threats. The threat analysis of usage control system in D1.1 identifies specific threats against the policy evaluation process. A policy infrastructure that supports policy evaluation must implement measures to mitigate possible threats. In particular the policies, metadata, and key stores must be protected against unauthorized modifications. Also, the association of metadata and policies with protected data should be protected. If an attacker is able to compromise either of these, then the attacker will be able to make the policy enforcement pointless.

2.1.1.7 Non-Repudiation

We have discussed requirements for authorizing users to access data and enforcing obligations. Despite designing the authorization and enforcement processes with utmost care, data misuse or leakage via unforeseen circumstance happen. Investigations are then needed to determine who accessed the leaked/misused data, and how a malicious user obtained authorization to access that data, or why obligations were not fulfilled. When a data sharing agreement is violated, the data sharing agreements between organisations such as research institutes or internal data management can require specific compensating actions to be performed. These can be monetary retributions, changes in the co-operation plans (e.g. new restrictions to data access for the violator) or internal disciplinary actions (suspension, sack, downgrade etc). In such situations, the data consumer should not be able to repudiate its usage history. As in the following example from (Medical Research Council, 2003) high-level data management policies require some entities to be burdened with the accountability of data disclosed:

Example 14. *"Each individual entrusted with [...] information is personally responsible for their decisions about disclosing it."*

2.1.1.8 Data Protection Requirements

While protection of sensitive content is primary responsibility of the enforcement layer, the policy infrastructure should support advanced features in addition to expression of basic authorization rules.

Derived Data

Data sharing agreements often want to specify usage restrictions over data derived from originally shared data. They may want to enforce same protection requirements on any derived data or may allow additional privileges for sharing data fragments. The policy infrastructure should be able to apply the policy requirements to derived data or data fragments copied from protected data. Following is an example requirement from the Sensitive Scientific Data test bed that also tells us about motivations for requiring protection for derived data.

Example 15. *"an ability for the system to track derived data and provides a 'provenance trail' would be invaluable. This trail provides evidence for the thorough conduct of the research and demonstrates the quality of the results as well as ownership. These are important elements for patent application, journal review and research assessment processes."*

Document Parts

In some applications, it is necessary to control access to parts of a document, rather than just controlling access to the document as a whole. Following is an example of such use case from the crisis management test bed.

Example 16. "*A TSO (Tactical Situation Object) shall be divided into a set of separate elements, and shall have different access policies for each element. Access to a given element is permitted only where the access policy permits it and not otherwise*."

Therefore, the policy infrastructure must be flexible enough to support requirements to protect parts of a document, as well as whole document.

Data Transformations

Data transformations can perform functions like data filtering and data enrichment. Sensitive data often needs to be filtered or removed before disclosing requested data set. On other hand, additional data or metadata may need to be added to prevent misuse or misunderstandings. We now discuss these functionalities in further detail.

Requirements for filtering data are often expressed in terms of minimum disclosure. The principle of minimal disclosure is a guidance rule requiring that only the minimum quantity of data necessary for a specific purpose be disclosed. The following policy from (Medical Research Council, 2003) is an example of such a requirement:

Example 17. *"The infringement of confidentiality must be kept to a minimum. Even when there is a strong justification for the study, the design must minimise the volume and sensitivity of the personal information that is disclosed, and the number of people who have access to it".*

In co-operative scenarios, data that must be shared between different organisations are usually disclosed so that only the results of an activity are shown, not the processes that led to them. With this strategy organisation can share their results without sharing their secrets, thus minimising any risk of information leakage.

A specific application of the data filtering is anonymising data, i.e. the process of deleting references to personal contacts and identity information in a data set. Anonymising data is only required when data refers to individuals. Similarly, other information such as images could be offered at different grades of resolution. Purpose of such transformation is to avoid the data subjects any trouble due to their data disclosure, as in the following example from (Medical Research Council, 2003).

Example 18. *"Researchers must also have procedures in place to minimise the risk of causing distress to the people they contact in the course of their research. [...] Information should be modified so that some, or all, of those who might see it are not aware of individuals identities".*

Requirements for enriching data set require additional information to be attached. For example copyright legislation states that authors of copyrighted assets must always be cited even if they are not the royalties' owners. In the research field (BBSRC), public data sets or committees require data to be released with additional information aimed to prevent misuses or misunderstandings. Thus, complying with policy may require adding information to data or metadata.

Example 19. *"Data, where appropriate, should be accompanied by contextual information or documentation (metadata) to provide a secondary user with any necessary details on the origin or manipulation of the data".*

Data transformation requirements raise many domain and application specific issues. For example, How is it possible to determine whether data is relevant or not for a given purpose? How is it possible to distinguish whether certain data that can be linked to a person's identity? The simplest solution to these problems is represented by metadata and labelling systems to organise and classify information. From the perspective of policy infrastructure, the act of data filtering and enrichment can be abstracted as functions that transform the data before it is disclosed to the end-user. A policy language can specify which data filtering functions should be executed and under which conditions.

2.1.1.9 Delegations

Delegation is the act of giving one's permissions or obligations to someone else. Usage restrictions on a file may allow or disallow the recipient to delegate its privileges to someone else. The subject who gives permissions is called a *grantor*, and the subject who receives the permissions is called a *grantee*. A grantor may delegate only those permissions that he/she possesses, and can retain the permissions for his/her own use. Need to delegate privileges sometimes arises in data sharing environment.

Example 20. *"in post incident investigation of a crisis management scenario, the organizations should allow extensive access to the investigators, and the leading investigator should be permitted to delegate temporary access rights to staff in their own organization".*

Thus, in Consequence project; there is a need to express policies that allow a subject to delegate certain permissions to another subject.

2.1.1.10 Policy References

A main characteristic of the examined data management and privacy practices is represented by references to policies located in different policy documents. Most of the examined policy documents refer to legislative acts or to policies followed by other organisations, as in the following example from (CCLRC, 2002):

Example 21. *"The storage and use of computerised [and certain manual] records relating to individuals is governed by the Data Protection Act (DPA)'98 ...".*

Policy references and chains are the consequence of the hierarchical structures usually organising different entities. The entities on top of the hierarchy (e.g. legislative and national committees) define a set of policies the low-level entities (e.g. private companies or final clients) must adhere to. Thus, data and resources managed by low-level entities are also subject to the policies defined by top-level entities. The policy infrastructure should be able to resolve references to any external policies and combine results of their evaluation.

2.1.1.11 Privacy Protection

Users accessing data may not belong to the data provider organization. Such users may view disclosure of their identity or personal information to an external identity for the purpose of policy evaluation as a compromise of their privacy. For example, access to certain data may require proof of residency in certain area/country. A user may have a proof of his/her address but may not want to disclose it to the data provider. In such case, the user needs to find an intermediate certifying authority that can verify user's credential and issue a certificate that is acceptable to the data provider.

Example 22. *"Existing Cabinet Office guidance, ... as an aid to emergency planning, response and recovery. This guidance is intended to provide a framework within which personal information can be used with the confidence that individuals' rights to privacy are respected".*

2.1.1.12 Document Specific Policies and Policy Confidentiality

In addition to the data sharing policies and organizational policy, creator of a document may want to protect a document with policies specific to that document-instance. To illustrate the need for instance-specific policies, we now present an example of data sharing in a crisis-management scenario. Consider a situation with natural disaster, to which several agencies respond. Lets say Alice is injured and is provided initial care by Red Cross. However, Red Cross needs to transfer Alice to a major hospital outside the disaster area. In this scenario, Red Cross should also share medical data of Alice with the Hospital. When a patient is treated by a medical facility, the patients can typically specify policies specifying who can access their medical record. For example, the patient may specify whether his/her medical records can be accessed by family members, or by other companies for advertising purposes. When Alice's medical record is shared, any such policy specified by Alice should be attached to the shared medical record.

The need to use document specific policy may also arise due to the need to provide partially offline access in certain scenarios. Consequence project needs to address situations when data may be accessed from thin client devices or portable devices, and the network availability is not guaranteed. In such cases, the availability of an external policy store is not guaranteed. Due to resource constraints of device, it may not be feasible to replicate entire policy store, or evaluate the entire policy set (as in policy store) on these devices. However, it may be possible to pre-evaluate the policies, and attach the applicable policies with the protected document. In other words, the attached policies are specific to the document instance. Following is a statement from the crisis management test bed expressing need for providing access control on portable devices.

Example 23. *"it is safe to assume that systems will typically control access based on the user's identity. This application could be running on a PDA used by a tactical command officer or on a desktop PC used by a control room officer"***.**

In certain cases there may be need to keep these policies confidential. For example, when dealing with extremely sensitive data it may be necessary to hide identity of users allowed access to that document. It may be noted that Consequence project does not aim to define specific mechanism for sending document from one part to another. The aim is to develop general mechanism that can be used for any of the specific data transfer mechanisms, like email, ftp, or data transfer on disk. A policy sent in plain text, is an unnecessary disclosure of information and should be avoided when dealing with sensitive information. An adversary who may view a policy gains knowledge about the system that he may not otherwise have. In example of medical policies given above, unnecessary disclosure of policy attached with a medical record can be considered as violation of patient's privacy. Therefore, it is desirable that policy infrastructure supports mechanism that allows use of document specific policy and allows keeping these policies confidential.

2.1.1.13 Revocation and Policy Modification

Revocation is the act of withdrawing permissions given to a subject. A data provider may want to take a decision to revoke access or modify the access criteria even after dissemination of data. The policy infrastructure should check for policy modifications and access revocation. This cannot be done if the recipient remains disconnected.

Data sharing policies may also need to be modified after they have been deployed. The policy infrastructure should be able to provide mechanism to support such policy updates, without major hindrances to day-to-day use of data. Following is an example of such required from sensitive scientific data test bed:

Example 24. *"In view of the relatively long timescale of collaboration, it is expected that the initial set of policies refined from data sharing clauses in the different agreements will need to evolve in line with the project ecology. For instance, change of personnel, new data sharing requirements for emerging results not covered by existing agreements. ... After consent has been obtained, new low level policies will be generated and deployed."*

In Table 1, we summarize the high level requirements for policy infrastructure in the consequence framework.

Requirement		Section
PDR 1.	Attribute based description	2.1.1.1
PDR 2.	Trust relation for obtaining attribute values	2.1.1.1
PDR 3.	Trust relation for evaluating security policy	2.1.1.1
PDR 4.	Authorization based on third-party approval	2.1.1.2
PDR 5.	Threshold based authorization	2.1.1.2
PDR 6.	Support for controlling manual dissemination	2.1.1.2
PDR 7.	Access based obligations	2.1.1.3
PDR 8.	Event condition based obligations	2.1.1.3
PDR 9.	Evaluation of conditions	2.1.1.4
PDR 10.	Deterministic evaluation	2.1.1.5
PDR 11.	Partially offline evaluation	2.1.1.5
PDR 12.	Threat mitigation	2.1.1.6
PDR 13.	Non repudiation	2.1.1.7
PDR 14.	Protection of derived data	2.1.1.8
PDR 15.	Protection of document parts	2.1.1.8
PDR 16.	Data transformations	2.1.1.8
PDR 17.	Delegation	2.1.1.9
PDR 18.	Policy references	2.1.1.10
PDR 19.	Privacy protection	2.1.1.11
PDR 20.	Sticky policies	2.1.1.12
PDR 21.	Policy modification	2.1.1.13

Table 1. Requirements for Policy Infrastructure

2.1.2 Metadata Generation Requirements

We identify in the following sections a set of requirements that the metadata generation component will need to satisfy. These requirements are summarised in Table 2 below, which contains references to the specific sections in which each requirement is further expanded on.

Requirement		Section
MDR 1.	Data Integrity	2.1.1.1
MDR 2.	Metadata Interoperability	2.1.1.12
MDR 3.	Metadata Derivation	2.1.1.13
MDR 4.	Root Metadata Traceability	2.1.1.24
MDR 5.	Data Fine-grained Access and Usage Control	2.1.1.25
MDR 6.	Data Usage Reporting	2.1.1.26
MDR 7.	Quality Management	2.1.1.37
MDR 8.	Offline Operability	2.1.1.48
MDR 9.	Conflict Classes	2.1.1.59

Table 2. Requirements for Metadata Infrastructure

2.1.2.1 Data Integrity

Data integrity is often at the top list of the security requirements for releasing scientific data in an open environment, even more so than confidentiality of data usually achieved through embargo-till-some-date policies. When data are held in the local storage server at some organisation, the server will have control on the access and usage of that data in a manner that ensures their integrity. However, in a Consequence scenario, data will be travelling to users outside the administrative domains of their originating organisation. In the current state, the local server will have to *trust* the user in a foreign domain that the latter will not tamper with the data in a manner that corrupts that data or any of its provenance, scientific and other information expressed as metadata. Such tampering has implication on the reputation of the data originator as well as quality management, and possibly, scientific consequences.

Example 25. In Table 3 of D6.1, we quote the third condition of the first Facility Data Policy *"No one can update or delete experimental raw data files, except the ICAT superuser."* We also quote the CSMD requirement MD8 *"The model must include redundancy information about the data. This is particularly important to preserve the data integrity."*

Therefore, the metadata generation component must ensure that the metadata generated will carry enough information for integrity-preserving policies to allow those policies to be evaluated and enforced. In its simplest form, such information could be an encrypted checksum of the data and its scientific metadata.

2.1.2.2 Metadata Interoperability

The metadata must be comprehensible to different cross-domain enforcement points, i.e. interoperability of the infrastructure must be respected with respect to the metadata being passed around. This requirement arises from the scenario where different scientific facilities decide to share their data by allowing users of a facility to search for, access and use data generated by another. Therefore, data will be travelling across different administrative domains; hence, it becomes a requirement that any metadata generated and attached to the data by some scientific facility (including any security-related metadata) be *comprehensible* to the policy decision point(s) of other facilities, within the assumption that all decision points

will be adopting the Consequence enforcement layer. It is also assumed that the local meaning of the global metadata will be agreed beforehand at each policy decision point.

Example 26. We quote business requirement TR12. Of D6.1 *"The data sharing policies are enforceable not just over data held centrally by the facility ICAT, but also when the protected data is disseminated and analysed on third party locations."*

2.1.2.3 Metadata Derivation

Scientific data may be derived from other *raw* data using a variety of analysis tools and scripts some of which are proprietary developed by the scientists. This introduces the question of what type should the new data have? The metadata for the new data may be generated at either the originator's or the receiver's domain. There could be any metadata inference system used by the recipient for generating the scientific metadata.

Example 27. We quote here the business requirement BR10. of D6.1 *"It should be possible to propagate data sharing restrictions from a source dataset to the derived data."*

2.1.2.4 Root Metadata Traceability

The root metadata, i.e. the metadata of the original raw scientific data, must always be extractable from the sub-metadata of any derived scientific data resulting from the application of analysis tools. The main motive of this requirement is to establish *impact factors* for the data generated at the scientific facilities. Such raw data may have contributed towards interesting and ground-breaking results, so, it is of importance to the originator to be able to trace the origins of the results to their data.

Example 28. We quote here from Section 2.5.1. of D6.1, page 17 *"Our test bed focuses on scientific research that gives rise to IP. In this context, an ability for the system to track derived data and provides a 'provenance trail' would be invaluable. This trail provides evidence for the thorough conduct of the research and demonstrates the quality of the results as well as ownership. These are important elements for patent application, journal review and research assessment processes."* And also the business requirement BR11 of the same deliverable *"It should be possible to identify the source of an inherited enforceable policy of a derived dataset."*

2.1.2.5 Data Fine-grained Access and Usage Control

The metadata must be expressible enough in order to accommodate information stemming from the clauses of the DSA under which the data is to be shared. This implies that the metadata must be expressive enough for fine-grained access and usage control policies applied to that metadata to be evaluated and enforced. For example, a DSA may specify that data originated in some scientific experiment cannot be analysed in certain prohibited countries or that the data cannot be read more than five times in each application slot, and where a slot is determined by some payment made in advance by the application. For such an example policies, the metadata must contain information on *place* and *date* of access and usage, as well as information on the number of accesses per data per application slot.

Example 29. We quote here the technical requirement TR10 of D6.1 *"The proposed policy infrastructure has the capability to actively monitor on-going environmental parameters to support context-aware data usage."*

2.1.2.6 Data Usage Reporting

The metadata may be needed for logging information about the usage of data. So, the metadata must be somehow mutable. Such mutability facilitates the fine-grained access and usage control requirement as mentioned in 2.1.2.5 above. Nonetheless, the mutability of some metadata is in fact a delicate requirement, in particular within the context of distributed and collaborative environments, for obvious synchronisation reasons. Moreover, the type of information logged by some policy decision point in the distributed system may differ from others. So, there must be an agreement by all parties on the exact metadata that needs to be mutable, i.e. on the ontology of the mutable metadata.

Example 30. We quote here Table 3 of D6.1, the 3rd condition of the 4th Data Sharing Schedule in Collaboration Agreement *"an audit trail must be kept."*

2.1.2.7 Quality Management

The metadata may occasionally give an indication of the quality of security and trust of the applications permitted to access the data at hand. This information could be in the form of conceptual security protection levels (e.g. low, medium or high) or in the form of some quantitative trust measure (e.g. a number in the range of 0 to 100). However, this requirement is challenging in the sense that it assumes that the policy enforcement point is aware of the security and trust level of the application, which is possible in certain standard applications and more difficult to assess in proprietary applications.

Example 31. We quote here Table 3 of D6.1, the 9th condition of the 1st Facility Data policy *"The facility would like researchers who have downloaded publicly available data to permit the periodic collection of usage information for data management purposes."*

2.1.2.8 Offline Operability

The metadata must indicate whether the usage and access of data must necessarily be controlled online through connecting to a site-wide PDP, or whether it is acceptable to use and access the data in offline mode directly on and locally at the user's machine. This implies that the metadata must also contain enough information about the data in case there are network failures.

Example 32. We quote here Section 2.5 on Scenario Use Cases in D6.1, page 15 *"Off-line – disseminated data will be used on a standalone machine without network connection. The properties of the data policies will determine if and how the data may be used."*

2.1.2.9 Conflict Classes

In certain cases, data (metadata) released under one DSA may *interfere* in the security properties of data (metadata) released under another DSA. For example, releasing filtered medical records under a DSA with a pharmaceutical company may be acceptable for the general goal under which the DSA was agreed. However, combining those records with missing personal data, which could have been released separately under another DSA, may

threaten the confidentiality property of the personal medical data. Such a scenario introduces the requirement of having *classes of conflict* among different scientific and other data. These classes will be based on identifying the classes of security threats to the data and can be useful in detecting conflicts in DSAs.

Example 33. We quote here the business requirement BR2., D6.1 *"A single party may have multiple agreements with different parties on the same dataset."*

2.2 Priorities

Table 2 illustrates our classification of policy infrastructure requirements. We have classified the requirements as basic requirements, intermediate and advanced requirements. The basic requirements comprise of essential features like interoperability, ability to express authorizations, obligations, and contextual conditions. The more advanced requirements comprise of nice to have features but are not essential in many scenarios.

Our highest priority will be to fulfil the basic requirements for data sharing and usage control. Our next objective with almost same priority is to fulfil essential requirements to mitigate the most-likely threats as identified in the threat analysis. Development of advanced features requires development of basic functionality first. Therefore, we consider them to have lower priority and in our work we hope to work on the more advanced features in later stages of the project.

Shared data may be in form of documents, data fragments from relational databases, XML fragments, or data stream. In Consequence project, we give priority to data being shared in the form of documents. We believe this approach lets us focus on solving essential problems of data sharing agreements and usage control. Our approach can then later be extended for data fragments or other formats.

Basic Requirements
PDR 1. Attribute based description
PDR 2. Trust relation for obtaining attribute values
PDR 3. Trust relation for evaluating security policy
PDR 4. Authorization based on third-party approval
PDR 5. Threshold based authorization
PDR 6. Support for controlling manual dissemination
PDR 7. Access based obligations
PDR 8. Event condition based obligations
PDR 9. Evaluation of conditions
Intermediate Requirements
PDR 10. Deterministic evaluation
PDR 11. Partially offline evaluation
PDR 12. Threat mitigation
PDR 13. Non repudiation

	PDR 19. Privacy protection
Advanced Requirements	
	PDR 14. Protection of derived data
	PDR 15. Protection of document parts
	PDR 16. Data transformations
	PDR 17. Delegation
	PDR 18. Policy references
	PDR 20. Sticky policies
	PDR 21. Policy modification

Table 2. Priority categories for policy infrastructure requirements

3 State of the Art

3.1 Models

Controlling usage of data after its acquisition or transmission to others is a requirement common to many applications and the focus of research work in many areas of computer science. In addition to research work on policy languages and framework for usage control, this survey draws from multiple research areas including:

- *Access Control* that focuses on specification and evaluation of policies for controlling access to documents and resources. Access control frameworks focus on expressing authorization constraints, contextual constraints, and delegation rules.
- *Policy-Based Management* that focuses on the declarative definition of rules constraining system behaviour. Their main advantage is that policies can be dynamically loaded or removed from the system to affect its behaviour without interrupting its functioning or changing its functionality.
- *Privacy* that focuses on the handling of information relating to individual persons. Issues addressed within this category include restricting the communication of and access to personally identifiable information based on the wishes (and possibly consent) of the patient, Data Protection legislation or both. Proof of compliance with the legislation is also an issue to be considered.
- *Enterprise Rights Management* (ERM) that focuses on the distribution of commercially sensitive information within and between cooperating organizations. Access to and usage of such resources can be subject to a wide set of different conditions depending on the particular scenario and data type. Typically, protected resources are intended for use in highly controlled environments like a company and its offices.
- *Digital Rights Management* (DRM) that focuses on the distribution of copyrighted material and in particular music and movies. Access to and usage of such resources is usually subject to some form of payment and thus models in this category focus on payments as the main condition for access.

3.1.1 Policy Languages and Models

3.1.1.1 $UCON_{abc}$

Park and Sandhu (Park & Sandhu, 2004) have proposed a family of usage control models called $UCON_{abc}$. These models address the core concepts of usage control, i.e. Authorizations (A), oBligations (B) and Conditions (C). All other issues like delegation and administration are considered second-order and have been left for future work. The models consist of eight core components: subjects, subjects attributes, objects, objects attributes, rights, authorizations, obligations and conditions. Attributes are divided in two sub-categories, i.e. mutable and immutable attributes. Mutable attributes are defined as attributes whose value can change as a consequence of access, while immutable attributes can only be changed by administrative actions (e.g. explicit intervention of an entity who has been given right to manage other entities). In $UCON_{abc}$ the authorizations are evaluated based on subject attributes, object attributes, requested rights and authorization rules. Obligations are interpreted as requirements that have to be fulfilled by obligation subjects for allowing access. Conditions are environmental or system requirements that are independent of subject and object attributes. The family of $UCON_{abc}$ models is categorized into sixteen core models based on: decision factors, continuity of decision, and mutability. Decision factors for usage control are authorizations, obligations, and conditions. Continuity of decision means whether the decision is computed before (pre) or during (on) the time when right is exercised. Mutability allows update of subject and object attributes at different times, before, during, or after the right has been exercised.

3.1.1.2 IBE based Sticky Policies

Mont et al. (Mont, Pearson, & Bramhall, 2003) have proposed an approach to implement the *sticky policy* paradigm. Their proposal allows data owners to control information even after dissemination. Important features of this model is that the usage policies can be attached to data in a temper resistant manner, and the usage model forces the requestor to communicate (get audited) with a trusted authority. The mechanism is based on Identity Based Encryption (Boneh & Franklin, 2001) and Trusted Computing Platform Alliance (TCPA, 2001). The TCPA integrity checking mechanisms can be used to check that the receiver's platform is a trusted computing platform, and it is conformant with the disclosure policies. IBE is used to attach the policy to the document as an "encryption key" and thus detachable only by the trusted authority.

Four algorithms specify the Identity Based Encryption (IBE) encryption scheme:

- Setup: the algorithm generates a set of public parameters and a secret master key kept by a Private Key Generator (PKG).
- Extract: using the public parameters and the master key the algorithm generates from an arbitrary string (used as public encryption key) a decryption key.
- Encrypt: using the public parameters and an arbitrary string the algorithm generates from a text the correspondent cipher-text.
- Decrypt: using the public parameters and a decryption key, the algorithm extracts the original text from a cipher-text.

Figure 1 shows how sticky policies through IBE are realized in (Mont, Pearson, & Bramhall, 2003). Although the focus is on private data protection, the same architecture can be easily extended to a more general scenario for data dissemination.

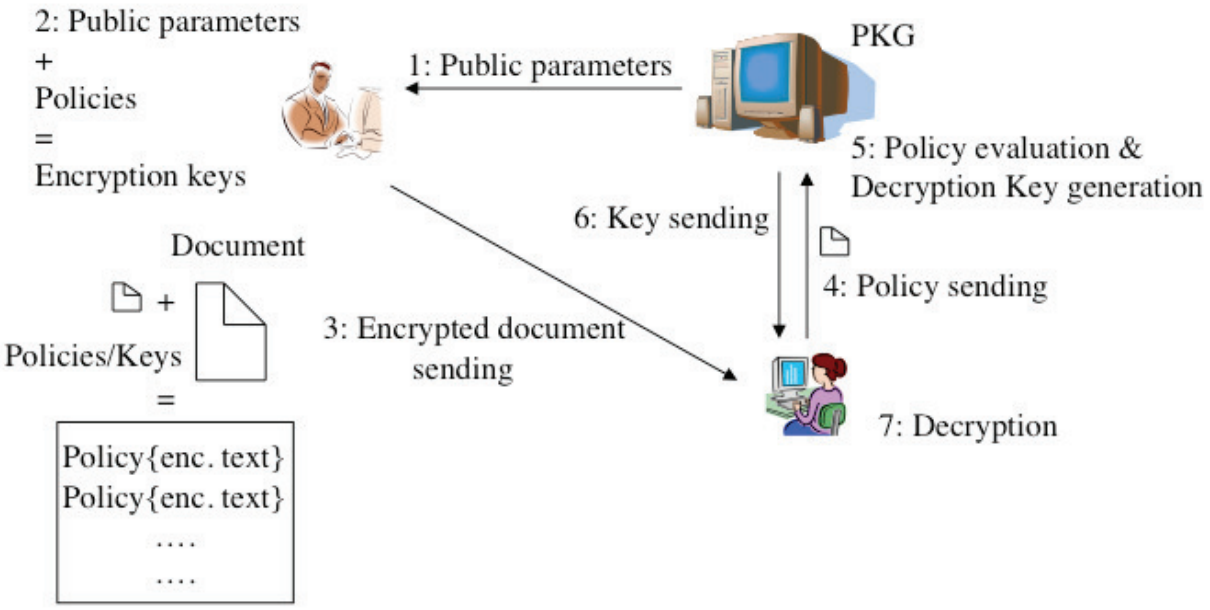

Figure 1. Sticky policy implementation with IBE

Let us assume the existence of a trusted authority (TA) playing the PKG role and issuing the IBE public parameters. Each section of the sensitive document is encrypted with a specific policy for protection. Cipher-texts and policies are then associated in an XML structure. The TA works as Policy Decision Point. Access requests for different document sections must include the correspondent encryption keys/policies so that the TA can evaluate them and generate the needed decryption keys. No access can be granted if the recipient can't supply the attached policies. This ensures that data is disclosed after the correct policy has been evaluated.

The main drawback of this architecture is the difficulty of realising off-line access. Users must be connected to the network to contact the TA and obtain the decryption keys. Moreover, the author assumes that each entity in the system is running on top of a TCPA platform. Thus without TCPA security cannot be achieved. TCPA also allows the TA to make checks over the clients' local environments with a certain degree of trust. Otherwise, the deployment of the Policy Decision Point on the TA would exclude clients' local environments form the set of specifiable access conditions.

The proposed architecture also assumes that policies can be distributed as clear text, unprotected. Some situations however could require policy secrecy, e.g. for user-specific rights. Finally, another issue is raised about the TA's location. From the details provided in (Mont, Pearson, & Bramhall, 2003), it is not clear where the TA should be deployed, how users know such location and how they can trust it to correctly manage their credentials.

3.1.1.3 Policy Description Language

PDL (Lobo, Bhatia, & Naqvi, 1999) is a policy language for specifying event-condition-action (ECA) rules aimed at network management applications. PDL was not designed for security purposes and does not include security aspects. However, it is a representative example of modelling ECA rules. The language is based on two kinds of statements. The first is used to specify how the system should react to a specific event occurrence and is expressed as:

event **causes** action ($t_1 = v_1$, ..., $t_k = v_k$) **if** condition

The second kind of statement is used to define new events (termed policy defined events) that should be raised when other events occur. These statements have the following syntax:

event **triggers** pde ($t_1 = v_1$, ..., $t_k = v_k$) **if** condition

Actions and events have typed arguments that can be used in the expression of the policy conditions. Thus it is possible to specify that certain actions should be performed only when the triggering events have particular attribute values. Primitive events can also be combined to create complex ones using conjunction and disjunction operators. In particular event composition rules are based on the concept of Epoch and event history. An epoch is a time window where multiple events are considered to occur simultaneously. An event history is a stream of event instances and is thus constituted by several epochs.

PDL represents an example of event-based policy specification language. The concepts of epoch and event history, as well as the presented composition rules, offer a good means to trace system behaviour and manage it. The same ideas could be exploited for usage control where composition of several permitted accesses; actions or general events over the time could generate a risk for resource security.

3.1.1.4 Ponder2

Ponder2 (Twidle, Lupu, Dulay, & Sloman, 2008) is a declarative, object-oriented language for the specification of management and security policies. It has not been designed for usage control but the language offers a wide range of interesting model of concepts that are worthwhile to explore. These include a hierarchical data structure for organizing managed objects, a policy evaluation points (PEPs) scheme, and their conflict resolution strategy.

A core concept of Ponder2 is that managed objects (resources) are explicitly grouped in a hierarchical domain structure. The managed objects can be added to one or more domains. The grouping is explicit and does not depend on an automatic attributes evaluation. Managed objects can interact with each other through procedure calls or can trigger events. Policies are used to control objects mutual interactions through authorization rules (i.e. permissions on procedure calls) and ECA rules (i.e. procedure calls triggered by events). A specific scripting language called PonderTalk and a Java programming interface allow to dynamical management of the tree hierarchy, specification of authorizations, and ECA rules.

Domains, events and policies are themselves managed objects, thus policy rules can be applied to them too (Note: policies written for domains propagate to domains members). This feature allows a policy writer to dynamically (and automatically with ECA rules) manage policies applied to groups of managed objects.

Within this architecture, principals, resources and services are all considered managed objects and contained in same domain structure. However, resources are not transferrable between remote domains. A managed object can't be moved to a remote domain with its state, attributes, content and applied policies. This limit is due to the fact that objects are seen remotely only through remote references integrated in proxies. This is why Ponder2 as it stands is not suitable for usage control.

The ponder2 policy language allows the definition of negative and positive authorisations and obligations. A particular feature of the language is the possibility to define the focus of an authorisation, i.e. the point of the interaction between service client and service provider, where the policy is evaluated. There can be policies permitting or denying the client object to invoke a service; policies permitting or denying the server object to accept an invocation;

policies permitting or denying the server object to issue certain replies after a service execution; policies permitting or denying the client object to receive certain replies.

Ponder2 uses the term “obligation” to identify event-condition-action policies. The policy language does not allow definition of obligation actions that must be executed before or after an authorisation. The following example policy shows an event-condition-action (ECA) rule syntax. The policy states that after a specific date a certain policy must be activated.

3.1.1.5 Rei

The main purpose of Rei (Kagal, 2002) is to support the development of context aware systems where contextual and environmental information is used as a support for services. The core idea of the system design is that every user is part of one or several domains. Domains can be both physical and logical and membership to one of them depends on context data gathered through sensors.

The Rei framework is composed of two basic components: Domain Server and Policy Server. The domain server interacts with sensors and other contextual data sources and decides users’ domain memberships. The policy server retrieves the policies for the specific domain and decides which rights, prohibitions, obligations and dispensations (actions not to be performed) are assigned to them. We focus our attention on the Rei policy language and not the underlying framework.

A policy object can be an instance of one of four policy types, i.e. right, prohibition, obligation and dispensation. There’s no difference in syntax between the four since they are always represented as condition-action tuples. Conditions can be based on context attributes, events and fulfilled actions. Actions are of the following form

action(ActionName, TargetObject, Pre-Condition, Effects)

and can be combined with four ”ordering” operators to create more complex ones:

- Sequence: an ordered set of actions;
- Non-deterministic choice: an action that is defined as the fulfilment of one or more of other actions;
- Iteration: an action defined as the repetition of an other one;
- Once: an action that can be performed only once.

Rei also offers special constructs to delegate, request, revoke and cancel rights at run-time. Only subjects having the right to delegate, as in the following example can execute delegation of rights:

has (subject , right (delegate (right (Action , Condition))))

Rei distinguishes between two different types of delegation, *when-delegation* and *while-delegation*. When-delegation requires the delegate to satisfy a specific condition at delegation time, before a right can be assigned forever. While-delegation requires the condition to be satisfied at all times after the delegation, for the right to be valid.

Rei offers policy writers a wide spectrum of specifiable policies. Both negative and positive authorizations (permissions and prohibitions), obligations, dispensations and delegation policies can be expressed on the base of different conditions. Context, attributes, roles, and even history-based conditions (through the assert statement and the ”ordering” operators) can be specified. Policy conflicts can also be managed with explicit meta-policies. However, Rei still lacks a construct for authentication policies, i.e., the authorization mechanism has no knowledge of how users are authenticated, and how the users are assigned to groups or roles.

3.1.1.6 OSL: Obligation Specification Language

The Obligation Specification Language (OSL) (Hilty, Pretschner, Basin, Schaefer, & Walter, 2008) is a policy language specifically designed to satisfy Usage Control requirements. The language is based on the concept of obligation formula, i.e. a condition that binds a usage-permission to different requirements. In particular the language considers two different types of requirements, i.e. usage restrictions (of the form: if condition then not usage) and action requirements (of the form: if condition then action). Conditions specify the circumstances enabling obligation formulas and are classified into five categories:

- Time Conditions: Action X must be performed within Z hours, Action X can be performed for Z minutes;
- Cardinality Conditions: Action X can be performed Z times;
- Event-defined Conditions: Action X can be performed only if event Z has occurred within that time interval;
- Purpose Conditions: Action X can be performed for purpose Z only. It is not specified how the purpose of an access request can be identified and matched with the specified condition;
- Environment Conditions: Action X can be performed only in an environment where Z is true.

Events (i.e. executed actions) can be organized in classes, thus allowing prohibitions on actions of the same class to be grouped all together. Events are also characterized by a set of basic parameters, including for example the subject and target of the executed action (no complex structures are allowed as parameters). Therefore OSL events are defined as the combination of a class with different parameters.

The following example defines the ”play” event member of the ”usage” class:

(play, usage {(Object, objID),(Device,devID)})

OSL’s semantics is defined over traces with discrete time steps. Each time step is associated with the set of events occurred in the correspondent time window. To cater for restrictions on duration and cardinality of actions, OSL introduces indexed events. Indexes are special attributes associating events to a time step. In particular the start index associates events to their starting time step while the ongoing index associates them to all the following ones. At each step of a trace, indexed events are associated to the currently executed action. In the following example the time step trace indicates that resource *m* has been played for three time steps (note: the time step length depends on the underlying system):

((play, {(Object, m),(Device, d)}),start)

((play, {(Object, m),(Device, d)}),ongoing)

((play, {(Object, m),(Device, d)}),ongoing)

The terms E_{fst} ($Event_e$) and E_{all} ($Event_e$) are used in OSL to indicate respectively the starting indexed event *e* and the whole set of subsequent events *e* (including the starting one). Accumulated traces can then be used to reason on policies and to verify the corresponding conditions.

The temporal trace is used as reference base for a set of operators (until, after, during, within) defining constraints on the events sequence. We will not treat in details these operators since (Hilty, Pretschner, Basin, Schaefer, & Walter, 2008) does not specify how conditions on the temporal trace can be used to trigger new actions. Though authors focus mainly on

authorizations, the idea of temporal traces triggering new actions and events is very interesting.

OSL offers four constructs to specify authorization policies:

- repuntil. Example: the movie m must not be played more than 3 times until a payment of 10 is made to r.

  repuntil(3,E_{fst}(play,(object,m)),E_{fst} ((pay,(currency,USD),(amount,10),(recipient,r))))

- repmax. Example: the movie stream s must not be played for more than 5 time steps.

  repmax(5,E_{all} (play,(object,s)))

- permitonlyevname. Example: the only usages permitted on the object oid are play and print.

  permitonlyevnam(play,print,(object,oid))

- permitonlyparam. Example: out of all send events with doc as the object parameter, only those where the recipient parameter has the value s_1 or s_2 are allowed.

  permitonlyparam(s_1,s_2,recipient,send,(object,doc))

In (Hilty, Pretschner, Basin, Schaefer, & Walter, 2008) authors describe how to translate OSL into subsets of XRML (Content Guard, 2001) and ODRL (Iannella, 2002). It is then possible to conclude that already existent enforcement mechanisms can be used to enforce OSL. Moreover it would be possible to use OSL as an intermediate language for interoperability.

OSL presents some very interesting features and concepts such as the idea of temporal trace and discrete time steps. However, its syntax and semantic is strongly based on Linear Temporal logic (LTL) (Pnueli, 1979), whose purpose is model representation and not policy specification. The result is a language with very complex grammar and syntax describing properties to be satisfied on a temporal trace. Properties represent policies to be enforced, but the approach looks unintuitive and allows definition of non-enforceable or meaningless policies.

3.1.1.7 SecPal

SecPAL (Becker, Fournet, & Gordon, 2005) is an extensible policy language for authorisation specification based on logical clauses. Access requests are treated as logical queries evaluated against a knowledge-base of pre-defined clauses. Predicates are used to assign rights to users with the further provision of a mechanism for right delegation.

A SecPAL authorisation policy is composed of a set of assertions of the following format: *A says fact if* $fact_1$ *,...,* $fact_2$ *,c*. Each assertion $fact_i$ specifies a property of a subject and is therefore composed of at least a subject identifier and a verb phrase. Verb phrases are of the kind: *can doAction [parameters], has access from [parameters] till [parameters], is attribute* and so on. Subject A is the entity asserting the specific fact, i.e. issuing the assertion claim. Facts can also be nested using the verb phrase *can say*, thus obtaining delegation of assertions. *c* represents a set of conditions constraining the values of the variables contained in the assertion. Negative conditions are also possible, thus allowing separation of duties, threshold and prohibition policies, as we will show later in the examples.

The main weakness of SecPAL is probably the fact that it is limited to authorisation policies and does not offer any construct to specify obligations, provisions, ECA policies and administrative (meta-) policies. In the following we will show some representative examples taken from (Becker, Fournet, & Gordon, 2005).

Parameterized roles, attributes and privileges can be obtained using verb phrases. In particular a *patient* parameter is used to express a value that change according to the specific access request.

NHS says x can access health record of *patient* if x is a treating clinician of *patient*

Separation of duty policies can be obtained by specifying methods that can be called by the resource guard component. In the following example, a payment transactions proceeds in two phases: initiation and authorization. The policy specifies the requirement that two different managers must execute these two phases. Also note that in this example, the permission to initialize payment has an additional history-based constraint, which prevents more than one manager being able to initialize payment for the same *payment* object.

`can-initiate-payment`(*requester, payment*):
`Bank` says *requester* is a manager,
not($^{\perp}$ *x* (Bank says *x* has initiated *payment*))

`can-authorize-payment`(*requester, payment*):
`Bank` says *requester* is a manager,
`Bank` says *x* has initiated *payment*,
$x \neq requester$

Threshold policies can be specified directly as in the following example, where the subject Alice is said to trust another subject if this one is trusted by at least three others.

`Alice` says *x* is trusted by `Alice` if

x is trusted by *a*, *x* is trusted by *b*, *x* is trusted by *c*,

$a \neq b$, $b \neq c$, a $\neq c$

`Alice` says *x* can say$_\infty$ *y* is trusted by *x* if *x* is trusted by `Alice`.

The verb phrase "can say$_\infty$" represents the possibility for the target subject of the phrase to further issue the assertion and thus represents a delegation. The symbol ∞ represents the unbounded length of the delegation chain, while to specify a one-step only delegation it is possible to use the verb phrase "can say$_0$". Delegation can also be constrained by restricting the parameters of the delegated facts or can be bound to a specific length of the delegation chain, different from 0 and ∞. The verb phrase *is a* allows to define attribute-based delegation and delegation chains bounded to a specific width, i.e. a chain whose intermediate delegators are constrained to have specific attributes.

3.1.1.8 Authorization Specification Language

Jajodia et al. proposed a Flexible Authorization Framework (FAF) (Jajodia, Samarati, Sapino, & Subrahmanian, 2001) that uses a logic based policy language called the Authorization Specification Language (ASL). FAF allows enforcement of multiple access control policies within a single system. FAF is a logic-based framework to express authorization requirements. Access control permissions or denials are derived by a sequence of applications of the authorization rules. These sequences include the propagation, the conflict resolution, the decision, and the integrity modules. In addition, it is ensured that every access request is either granted or denied, therefore ensuring completeness of the authorization policy.

The ASL syntax is built from constants, variables, and predefined predicates. The constants and variables range over authorization objects, subjects, actions, and roles. ASL includes the

following predicates to represent explicit authorizations (cando), derived authorizations (dercando), applicable authorizations (do) after resolving conflicts in derived authorizations, actions performed (done), overriding predicates (overAO and overAS), integrity violations (error), and for representing subject and object hierarchies (hie-predicates). Application specific predicates, called rel-predicates, are also allowed. Limitations of ASL include a lack of constructs to express obligations, requirements for ongoing access, threshold-based authorizations, and authentication policies.

3.1.2 Metadata Models

3.1.2.1 Generic Semantic Web Ontologies

DAML-S

DAML-S (Ankolekar et al., 2001) was designed as an upper ontology for describing Web services based on DAML+OIL (Connolly et al., 2001) with the objective of rendering Web services more ``computer-interpretable" and facilitate their discovery, invocation, interoperability, composition, verification and runtime monitoring.

DAML-S provides three main layers of ontologies: *Service Profiles*, *Service Modelling* and *Service Grounding*. At the top of the ontology, DAML-S provides the class **Service**. The subclass **ServiceProfile** provides information necessary for the discovery of the service by clients or other services. This information may include a description of the service, its provider, its functional behaviour and several functional attributes such as its type, degree of quality, geographic scope and communication throughput.

The second subclass is that of **ServiceModel**, which describes how the service works, i.e. it provides a model of the service. Usually, a service is modelled as a workflow process hence it also comprises process and process control ontologies with three classes of processes being distinguished: atomic, simple and composite.

Finally, the third class is **ServiceGrounding**, which specifies the details of how the service is accessed, i.e. the details of its protocol and message formats, serialization, transport and addressing. Naturally, such a detail was based on the Web Services Description Language (WSDL) (Christensen et al., 2001).

In (Denker et al., 2003), the authors propose an extension of DAML-S with ontologies that describe security properties and standards related to Web services. The ontologies presented include credentials such as smart and identity cards and X509 certificates as standardised by XML Signature (Mark Bartel et al., 2008) as well as security mechanisms such encryption functions, signatures and protocols. They also propose a reasoning engine based on the Java Theorem Prover (JTP) (Fikes et al., 2003) for matching security requirements of the user and those proposed by the service it requests.

OWL-S

Recently, and with the evolution of DAML+OIL into the new Ontology Web Language (OWL) (Bechhofer et al., 2004), DAML-S was replaced by the new OWL-S (Martin et al., 2004), a language based on OWL. There are some differences related to the representation of processes in collections between DAML-S and OWL-S. More precisely, in OWL-S, it is not possible to build process hierarchies at different abstraction levels and actual instances of processes are missing and do not represent actual events anymore.

Using OWL-S, the authors in (Kim et al., 2005) propose alternative security ontology called the NRL Security Ontology. NRL can be used at various levels of detail for annotating resources in general and not only Web services and furthermore, it is extensible, reusable and provides a higher-level mapping from security requirements to lower-level capabilities. NRL contains rich ontologies for describing credentials, security algorithms such as cryptographic functions, security assurances as well as service and agent security ontologies.

3.1.2.2 Semantic Web-based Ontologies for Policy Expression and Enforcement

Rei

Rei (Kagal, 2002) is a policy language that provides constructs for expressing deontic logic (von Wright, 1951). The language was defined for the Me-project, which follows the approach of dividing a distributed system into overlapping domains, each equipped with a context-aware policy that allows the domain to restrict the behaviour of entities within it based on information describing their properties or properties of their environment. Due to this dynamic nature of the project, Rei was developed as a general policy language that can be used to express not only security policies, but also management and conversation policies. In this sense, it is a generic language in that it is not tied to a specific domain such security, network management etc.

The essence of Rei is the deontic constructs of rights, prohibitions, obligations and dispensations. Each domain policy is composed of a number of *policy objects* that express whether an agent has the right, prohibition, obligation or dispensation to perform certain actions on objects. The policy object also contains environment conditions that must be true for the object to apply. Similarly, actions have a complex structure that contains the action name, the name of the object on which the action is being applied, a set of preconditions that must be true for the action to happen and an effect to express the effect of the application of the action. Policy objects can also express right delegations, right/action requests, right/action revocations and request cancellations. The language also contains meta-policies that can be used to interpret policies and resolve conflicts among them.

Associated with Rei is a policy engine that interprets Rei policies written in Prolog and RDF (Lassila and R. Swick, 1999). The RDF interface is based on a domain-independent ontology that allows the automated translation of policies extracted from other sources of information into the Rei policy language. Furthermore, in (Kagal et al., 2004), the authors propose the integration of Rei policies based on OWL-S (Martin et al., 2004) semantic Web descriptions.

Rein

Rein (Kagal et al., 2006) is a Web-based policy management framework that emerged out of Rei. In Rein, policies, languages and meta-policies can be combined and handled in the same manner any Web resource is handled. In fact, such resources and their policies form a *Rein policy network*, in which entities are located using Uniform Resource Identifiers (URIs) and communicated with using the Hypertext Transfer Protocol (HTTP). Rein forms the basis of the Policy Aware Web project (http://www.policyawareweb.org/).

The Rein framework consists of two parts: OWL or RDF-S (Hayes, 2004) ontologies that are used to describe policy networks and their access control and a reasoning engine, which acts as a Policy Decision Point (PDP) deciding whether a request is to be granted or not. Policies are connected using N3 logic (Berners-Lee et al., 2008).

Both Rei and Rein have two aspects that are interesting from the point of view of project Consequence: First expressing deontic concepts, which are an important element of digital rights management and second, context-awareness.

KAoS

KAoS (Uszok et al., 2004) is an ontology-based policy management framework for semantic web services. It uses OWL-based ontologies for defining policies. The KAoS Policy Ontology (KPO) defines at the most basic level the class Policy, which has as values Authorisation and Obligation Policies. These in turn are could be either positive or negative values. A policy also has environmental values such as the site enforcement, the time of updates and a priority level. Policies are used to control actions, which are kept in an event/action history. KAoS also extends KPO with additional ontologies for actors, groups, places and various actions and policies specific to particular environments and application domains.

The implementation of KAoS provides a number of functionalities including the bootstrapping of the KPO from a pre-defined configuration, ontology namespace browsing and management, domain and actor class creation, policy creation, distribution using a directory service, disclosure, analysis using Description Logic (Baader et al., 2003) and situation (context) awareness.

KAoS has been applied mainly to three application areas: policy management for Grid-computing environments, policy compliance verification for semantic web workflow compositions and policy enforcement during workflow executions. In Grids, an OGSA-compliant (http://www.globus.org/ogsa) version of KAoS was developed for managing fine-grained security policies for Grid services based on Globus (http://www.globus.org/). This management includes the registration of users and resources in the KAoS domain, the definition and enforcement of authorisation (and recently, obligation) policies using the concepts of actors, actions and context.

The DARPA DAML project Coalition Search and Rescue Task Support (CoSAR-TS: http://www.aiai.ed.ac.uk/project/cosar-ts/) is an interesting project that KAoS has been used in for the definition of policies constraining resource usage among partners in realistic and highly dynamic search and rescue coalitions. This project will be relevant to the "Managing Data Communication in Emergency Situations" scenario in project Consequence.

In the other two application areas, KAoS aims at reasoning on policies created for semantic web workflows and enforcing them. Such policies are concerned with action authorisations within the scope of the workflow within they are performed. The verification aspect is at planning or compile time, whereas the enforcement aspect is at runtime. Example projects for both areas are given in (Uszok et al., 2004).

3.1.2.3 Semantic Web Rule Languages

RuleML

The Rule Markup Language (RuleML) (Lee and Sohn, 2003) (http://www.ruleml.org/) was defined to encoding forward and backward deduction, rewriting and other inferential-transformational rules in XML. The language has evolved to RDF/XML-based RuleML and RDF-based RuleML. Of particular interest to Consequence is the initiative set forward by the Policy RuleML Technical Group (http://policy.ruleml.org/), which aims at defining a standard based on RuleML that will act as an interchange medium among the different policy languages. The focus in the new standard will be on deontic logic including permission and prohibition rules, duty assignment rule and empowerment rules.

SWRL

The Semantic Web Rule Language (SWRL) (Horrocks et al., 2004) combines both the languages of OWL and RuleML. The language extends OWL axioms to include Horn-like rules. Both an abstract syntax, which extends OWL's abstract syntax, and an XML-based concrete syntax, which combines OWL and RuleML XML syntaxes, are provided. Furthermore, a direct model-theoretic semantics are given to the abstract syntax based on interpretation rules.

3.2 Products/Technologies

3.2.1.1 XrML

The XrML language (Content Guard, 2001) was originally proposed as an open standard for expressing rights. XrML defines a basic data model for its core concepts that can be extended to support additional requirements of an application domain. The language's extensibility is realised through a set of core concepts built upon the XML Schema and pattern matching techniques. The set of core concepts defined in the initial namespace includes grants, licenses, principals, resources, rights and conditions. A grant element describes an authorisation and it has four child elements: principals, resources, rights, and conditions. A license contains a set of grants and is signed by an issuer. A license also contains additional information like its expiry date.

XrML focuses exclusively on authorisations. Obligations and delegations can be included as building blocks for licenses. However no construct is provided to specify constraints based on temporally ordered events or actions. Temporal conditions are simple time limits to rights wielding. The concept of time line, i.e. of temporal succession of events, states and actions is not considered. A significant limitation of XrML is the inability to define negation. Therefore it is not possible to distinguish unregulated actions from forbidden ones, and negative assertions can't be expressed.

3.2.1.2 ODRL

ODRL (Iannella, 2002) is an XML-based language to express access conditions over digital resources. The syntax of ODRL is simpler than syntax of XrML, but its basic concepts are very similar to those of XrML. Moreover, ODRL offers an extension mechanism to add new terms and types to the language’s existing set of terms and types. The basic elements of an ODRL policy are assets (resources), rights, and parties. Parties include both end-users and rights holders (i.e. originators or entities that can assert some form of ownership over the assets or its permissions). The explicit distinction between rights holders and users is justified with the provision of two constructs expressing rights offers and agreements. These concepts represent the interaction between users and rights holders to create licenses. The interaction however is very limited and users are only allowed to accept or deny offers. No real negotiation is possible. As a result, the agreement that is generated by an offer acceptance is simply a version of the offer tailored on the specific principal. It can be compared to a license token, i.e. an attestation of rights with some limiting conditions. For example, if an offer specifies that only employees of a certain company can access an asset, then the agreement resulting from the offer acceptance will include as principal the actual name of the employee who accepted in place of the reference to the employee group.

Authorization policies, i.e. granted rights, can be bound to constraints, requirements and conditions. Constraints represent limits to a right exertion, such as a limited number of usages or an expiry date. Requirements are the ODRL's equivalent for obligations. Finally, conditions are exceptions (depending on attributes values) that if verified cause the permissions expiration. Users', assets' and environment's attributes are specified through the "context" tag.

One of the main feature and yet problem of the language is that it strongly depends on the concept of identifier. Though universal identifiers notably ease the policy enforcement process, they require a stable and centralized infrastructure and identification system recognized by all the entities involved in the data exchange. The set of possible scenarios where ODRL could be exploited is thus restricted. As well as XrML, ODRL focuses exclusively on authorisations with the further lack that delegations can't be specified. In particular it also offers a construct to specify revocations, i.e. the invalidation of an established agreement. Unfortunately the language specification does not include any tip regarding who can revoke an agreement and under which condition. Furthermore, ODRL does not include negation as a basic element for policy specification.

3.2.1.3 XACML

XACML (Moses, 2005) is an XML-based standard specifying both an authorisation policy language and an access request-decision-response data-flow protocol and message format. Figure 2 shows a classical component deployment to perform the protocol. Each access request (containing information about the request subject, the requested resource, the requested action and the request environment) must be sent to a component protecting the resource called the Policy Enforcement Point (PEP). The PEP queries a Policy Decision Point (PDP) to get an access decision. The PDP can look for policies applicable to the request through the Policy Administration Point (PAP) and can request attribute values to a context handler if the request message does not contain all the necessary information. Once the corresponding policies have been evaluated, a response containing the access decision and a set of obligations to be performed can be returned to the PEP. It must be underlined that the presented diagram does not prevent from deploying multiple context handlers, PAPs, PDPs etc.

The policy language allows the specification of general access-control requirements and offers an extension mechanism to add new concepts to the pre-defined ones. Responses to access requests can include four different values: Permit, Deny, Indeterminate (indicating an error in the evaluating process or the need for more information) or Not Applicable (indicating the request cannot be answered).

At the base of any XACML policy is a *Policy* or *PolicySet* construct. A PolicySet can contain multiple Policy and PolicySet objects and even references to policies contained in remote

locations. This feature raises the problem of conflict resolution. To address this problem,

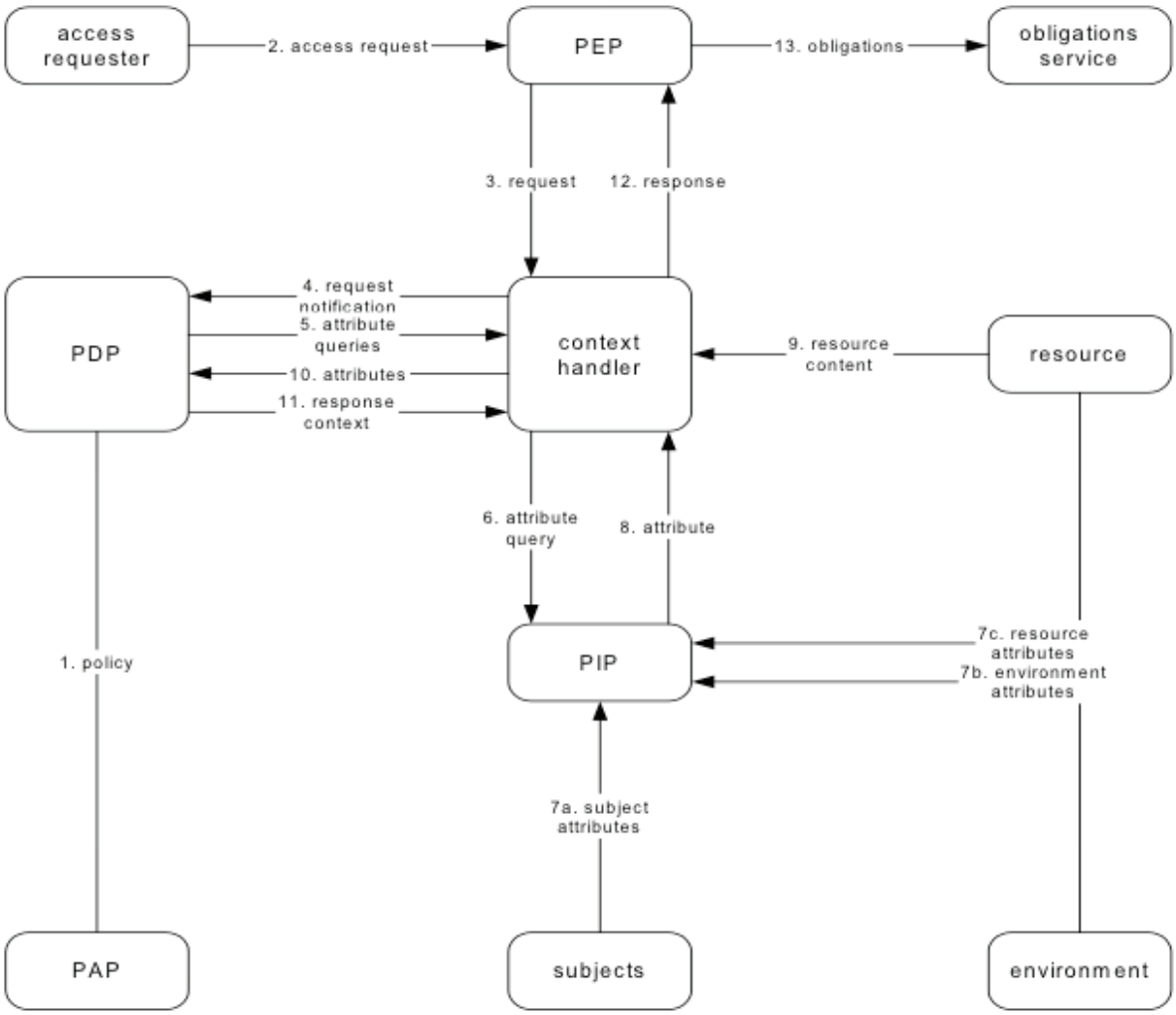

Figure 2. XACML data flow diagram

XACML offers a solution based on policy combining algorithms. A policy-combining algorithm can be specified for each policy set and that algorithm is used to solve conflicts that arise between policies contained in that policy set. Examples of combining algorithms are: Deny-overrides (if a rule or policy returns Deny, then the policy set returns Deny), Permit-overrides (if a rule or policy returns Permit, then the policy set returns Permit) and First applicable (the first policy applicable for the specific resource overrides all the others). Since each protected resource is univocally associated to a policy or policy set, conflicts between different policy sets are impossible. Policies also specify a target, i.e. a set of conditions on the Subject, Resource and Action that must be met for the policy to be applied to the specific request.

Policy evaluation is based on four different types of attributes: Subject, Resource, Action and Environment attributes. Each attribute can have an issue date and an issuer identifier. Policies can specify conditions, which may use constant values, range of values, and specific issuer identifier. Any attribute query returns a special attribute type called bag. A bag represents an unordered attribute set containing all the attribute values found by the attribute query. Functions are then used to compare attribute bags with the values specified in a policy. A simple function example is the [type]-one-and-only that returns an error if the bag contains more then one value, the only value contained otherwise.

XACML policies can also specify obligations. In XACML obligations are interpreted as provisional actions (Kudo & Hada, 2000) that must be performed before access is granted to the user.

Even if XACML offers very useful constructs and allows the definition of a wide set of policies, it also presents some issues. The first and most evident problem is the same afflicting almost all standard xml-based languages, i.e. complexity and verbosity. Secondly it does not offer any means to specify how policies can be modified at run-time or as an effect of events occurring in the system. It does not offer any construct to deal with time. It does not allow specification of temporary authorisations, i.e., permission is granted only while certain conditions are met. Neither does it allow specification of obligations that must be fulfilled after the access request, nor does it allow specification of temporal constraints over obligations.

3.2.1.4 EPAL: Enterprise Privacy Authorization Language

EPAL (Ashley, Hada, Karjoth, Powers, & Schunter, 2003) is an XML-based policy language developed for the purpose of enforcing user's privacy preferences when data is exchanged between enterprise applications. EPAL provides data handling control through fine-grained positive and negative authorisations.

Although web servers can publish their intended privacy policies, users have no guarantee they will actually comply with them. Data gathered by network and Internet services is accessed by applications underlying the simple web interfaces and is often moved from one company to another (e.g. business partners). P3P (Privacy Policy Language) and Appel (A P3P Preference Exchange Language) are two languages designed to give users capability to check what usage their data will be subject to. On one hand P3P allows web sites to publish their privacy practices policies, i.e. their intended use of the information they collect. On the other hand Appel allows a web surfer to specify which privacy practices he accepts. P3P (Privacy Policy Language) is a too general and high-level language to express enforceable policies (many rules are expressed with natural language) applications can comply with. What enterprises need is a policy language to express enforceable rules over data usage. EPAL is the policy language defined in this context.

The data gathered through a web site is usually exchanged between different applications. Therefore the first problem EPAL addresses is the definition of a common vocabulary all the entities exchanging data and policies could understand. While XrML and ODRL rely on name-spaces that can be downloaded at run-time to add new concepts to the basic ones, EPAL chooses to define a priori all the terms that must be known to access data. Only those entities that know and accept the defined vocabulary can satisfy the described policies. An EPAL vocabulary can be divided in different sections, each listing the terms of a specific category. In particular, it is possible to define entity categories (roles, individuals, organisations etc.), data categories, purposes categories (i.e. the reasons users want to access data for), actions categories, containers categories (i.e. context data and attributes) and obligation categories. Entity, data and purpose categories can be organised in a hierarchical tree where parent categories group all their children. After a vocabulary has been defined, policies based on its terms can be written.

EPAL allows policy writers to define a wide number of fine-grained authorisation policies. However, it just focuses on authorisations like XrML and ODRL. Moreover the types of policies that are possible to specify are quite simple. It is not possible to define conditional obligations, to combine conditions with boolean operators, and to use temporal operators (e.g.

sequence, once etc.) for obligations or past conditions. Even delegations are not considered at all. However, EPAL allows the definition of negative authorisations. While this possibility increases the language expressiveness, it raises the issue of policy conflicts.

Authorisation policies conflict when they are valid at the same time (both conditions are satisfied) and express opposite permissions for the same principal. A policy language whose policies can conflict must offer a means to solve risen conflicts. EPAL's conflict resolution strategy simply gives priority to policies according to the order they have been written. The proposed solution is too simplistic since it charges policy writers with full responsibility for conflicting resolution decisions. For very complex policies this could easily result in errors.

3.2.1.5 AuthorX

Author-X (Bertino, Braun, Castano, Ferrari, & Mesiti, 2000) is a tool for XML files protection and distribution. Its purpose is to realise Access Control over XML documents distributed through two different dissemination modes, i.e. pull and push modes. The former consists in a traditional direct request from a user to a central data repository. The latter assumes the subscription of different users to periodically changing documents. Dissemination is performed through a periodical push of the updated documents from the central repository to the client machines.

Author-X realizes protection at different levels of granularity exploiting the structured characterization of XML and DTD documents. Policies can in fact be applied to sets of documents satisfying the same DTD or to specific sections, subsection and attributes of specific documents. Authorizations are of the form (users; protection-objs; priv; prop-opt; sign) where the terms represent respectively: the subjects and the targets of the authorization, the access mode (read, navigate, write and append), the method of propagation of policies through the different levels of the XML tree (none, cascade, or just first level) and the sign of the permission (denied or granted). Policies are stored at the server side as an XML file (the "policy base"). Another XML file (the "credential base") stores the list of users and their credentials. Authorisation policies identify users through XPath expression over the credential base. Protected objects can be identified with different levels of granularity specifying a DTD (for protection of a set of files) file or an xml file. The target data set can be further restricted specifying an XPath expression identifying the single sub-elements to protect. In particular it is possible to specify the propagation option (CASCADE, FIRST-LEVEL, NOPROP) controlling if a policy applies to its target's direct or indirect child elements.

Since both negative and positive authorisations are possible, conflicts may arise between different policies. The language conflict resolution strategy assumes that more specific and negative authorisations prevail over the others.

The Author-X policy language provides a good means for multi-granularity protection. However, only authorization policies can be specified. No constructs are provided for obligations and authentication policies or rights delegation. In particular, credentials are stored and provided by the central server issuing the document. The authors do not consider a distributed system including different credential provider authorities.

3.2.2 Metadata Management Products

3.2.2.1 The Information Catalogue (ICAT) System

ICAT is a software suite that catalogues, in an automated manner, information related to the domain of scientific experiments based on the taxonomy defined by the CCLRC Scientific MetaData (CSMD) model described in Section 2.4.2 of Deliverable 6.1. Such information

includes topic indexing, provenance, data holding, any related material or experiments and access conditions, all of which are represented as a database schema.

ICAT offers a set of Web-based API, which allows the developer to interact with the catalogue in a programmatic way. The API offers functionality for the submission and retrieval of information as well as physical access to the data. It also offers access control based on simple RBAC-based policies. ICAT also offers a Web-based data portal through which ordinary users get access to the data and its metadata.

The main advantage of this software is that it is currently used by some of the scientific facilities in STFC, such as the STFC ISIS Pulsed Neutron and Muon source, the STFC Central Laser facility and the Diamond light source facility, in which STFC is partially involved. More description of this product is provided in Section 2.4.1 of Deliverable 6.1.

4 Proposed model/approach

In this section, we discuss the proposed policy infrastructure and metadata generation infrastructure. We identify their components and discuss their role in the overall architecture. We also present a policy language suitable for expressing data sharing and usage control requirements.

4.1 Policy Infrastructure

The main purpose of policy infrastructure is to evaluate requests for accessing resources. The evaluation of an access request may require a one-time access decision, or it may require continuous monitoring of certain parameters during the lifetime of the request. The result of evaluating an access request may also include a list of actions that must be performed before or after the access request to comply with the security requirements. The security requirements are expressed as a set of policy rules, which form a security policy.

More than one security policies may exist in an organization participating in data sharing. These security policies can be broadly categorized as organizational policy and data sharing policies. Policies are also often categorized as high-level policies and low-level policies. The high-level policies are specified using more general terms or abstract rules compared to the low-level policies. The low-level policies represent the concrete policies that are executable. In this work package, we always talk about executable policies and may not explicitly say so each time. There exists an executable organizational policy and one ore more executable data sharing policies in an organization.

Let us now briefly review the role of policy infrastructure in the overall Consequence framework. The data sharing policies processed by the policy infrastructure are a product of refining the data sharing agreement signed by an organization. The data-sharing infrastructure (WP2) is responsible for refining the data sharing agreement of an organization into the enforceable data sharing policies. The enforcement layer (WP4) depends on the policy infrastructure for evaluation of access requests. The policy-infrastructure in turn depends on the security token service for verification of user identity and attributes, and on metadata (object attributes) produced by the metadata-generation component. Before we can discuss further details of policy infrastructure, we must understand the different data sharing scenarios. While WP5 and WP6 consider the data sharing scenarios from the perspective of crisis management and protection of sensitive data, in following discussion we categorized scenarios based on how data is shared and in which security domain the sender and recipient access document.

4.1.1 Data Sharing Scenarios

The deployment of policies, policy evaluation and enforcement components depends on how data is shared. Data can be shared manually or by automated means. Typically, manual data sharing is done for low-volume of data, or when data needs to be shared on a per request/need basis. Examples of methodologies used for manually sharing data include sending a document by email to specific recipient, or copying a document onto a storage media like USB flash drive and giving that to the recipient. In automated data sharing, the sender and recipient both are software applications. This kind of data sharing is appropriate when large amount of data is shared on a regular basis.

In a typical data sharing scenario, the sender and receiver belong to different organizations are using IT resources (e.g., workstations, applications, servers, etc.) managed by their respective organizations. However, this is not always the case. For example, in the STFC test bed scenario it is possible that researchers from different organizations share IT resources, e.g., when they are working in same lab for a collaborative project. Also, note that the documents shared between two organizations, may be owned by neither of them, one of them, or jointly by both of them. In the case, when the document is not owned by either of the organizations, one of them just acts as a provider of the document to the other organization. The ownership of document can be a criteria used to decide, who can add or update policies on the document.

A data sharing agreement may specify different policies for users of participating organizations, and the event condition action rules applicable in their domains may also be different. During policy evaluation, the policy infrastructure must be able to determine which policies are applicable and which are not. From above discussion of data sharing scenarios, we can observe that for various functions, the policy infrastructure may need to know, which (organizational) domains do the sender and recipient belong, and who owns the protected document. The policy language must provide sufficient constructs to encode relevant information to make these kinds of decisions.

4.1.2 Protected Data and Metadata

In the threat model for usage control presented in D1.1, we recognized one of the threats as ability of an attacker to bypass a genuine policy enforcement layer and access the data directly. To mitigate such threats, the sensitive data must always be kept encrypted when stored. Only a genuine policy enforcement point should be able to obtain suitable decryption key when permitted by the policy decision point.

Enforcement of policies requires the policy infrastructure to know the protection requirements of the protected document. To evaluate the security policy, the policy engine needs to identify the object for which access has been requested. Encryption of sensitive data makes the process of determining the security requirements harder. To understand this problem, let us consider an encrypted document received by a recipient. The path of file where it is stored is irrelevant because the recipient could have saved the received document anywhere in his/her file system. Content-based rules cannot be applied because the enforcement layer is not yet able to decrypt the protected content. To overcome this problem, it is necessary to package metadata (as plain text or encrypted such that it can be read by enforcement layer before authorization) in the protected document. The policy evaluation method can use the metadata to determine the policy rules applicable to that document. It must be noted that attached metadata needs to be protected against unauthorized modifications. We assume the metadata is simply specified as a list of attribute and value pairs.

4.1.3 Policy language

In this section, we present a policy language to satisfy usage control requirements in data sharing scenarios. Our intention is to present the elements rather than final syntax. Existing policy languages suffer from drawbacks that make them unsuitable for usage control. Their scope is often too specific and can't express some of the concepts usage control requires, or are just too complicated for untrained users that are supposed to use them.

A data sharing policy primarily expresses constraints for accessing data and obligations. To express these constraints, a policy must be able to specify the protection objects (that a policy controls), actions performed on those objects, and subjects (whose actions over the objects are controlled). The objects and subjects must be specified in a way such that the specification can be interpreted in a consistent manner in both sender and receiver organizations. Use of unique identifiers should be avoided, as it requires additional infrastructure support their management and guarantee uniqueness of identifiers. A typical way to describe the objects and subjects is with *attributes*. This is the approach taken in our policy language. The selection of attributes strictly depends on the data model and data types the specific application scenario addresses, therefore the selection of object attributes is left to the policy writer. The subjects identified in the policy can be users, automated entities acting on behalf of users, autonomous entities etc.

In addition to expressing authorization and obligation rules, a data sharing policy must be able to express trust relationships, trusted authorities, services, and authentication requirements, as earlier identified in the requirements (Section 2.1.1). To support these features, our policy language needs to support declarative statements for credentials, certification authorities, remote services, and rules for describing authentication requirements for a subject to acquire a role.

We first define the concepts of credential (and credential types), certification authorities and remote services. A *credential* is an authenticated (e.g. signed) statement issued by an entity (the issuer) and typically asserting an attribute value or a past event/interaction relating to another entity. A *certification authority* for an entity is a credential issuer whose issued credentials are directly trusted by the entity itself. A credential is trusted by an entity if the entity believes the contained information is true and can be used in a decision and evaluation process. A *remote service* is an entity external to the local device offering trusted functionalities and resources.

A policy writer must be able to univocally identify remote services and authorities he/she trusts. The identification mechanism depends however on the specific underlying system. In the following we will only show a simple example using URLs but different notations and mechanisms could be used (e.g., public encryption keys, IP addresses etc).

In our language every credential is an instance of a credential type. A *credential type* is a template (i.e. format) any real credential must satisfy to be assigned to that specific type. The template is a simple set of attribute names representing the attributes values an instance of the specific credential type must contain. In the example below a remote service, a certification authority and two credential types (an employee credential and a trust credential) are defined:

```
RemoteService GSS = www.gss.co.uk;

CertAuthority ICL = www.icl.doc.ic.ac.uk;

Credential employee(Role, Name, Surname, Office);

Credential trust(TrustLevel);
```

Example 25. Remote service, Certifying authority, and Credential

Every credential presented by a subject and matching the definition in the policy (i.e. the set of contained attribute names) will be considered as a valid attribute container.

In the next sections we describe three different types of policies that address many of the presented requirements: authentication policies, authorisation policies, and event-condition-action (obligation) policies.

4.1.3.1 Authentication Policies

Authentication policies are rules specifying how unknown subjects that try to access a protected object can be classified and how their attributes can be authenticated and verified. The language we propose would allow a policy writer to specify how subjects can be assigned to different roles (i.e. groups) on the basis of the credentials presented. Roles are specified using a combination of one or more credential specifications. The credential specifications can be combined using OR and AND operators. To specify that several different credentials or different instances of the same credential type (but being part of different roles) are needed it is sufficient to list the different specification with the AND operator. The OR operator can be used to specify alternative credential configurations that lead to the same role assignment.

Each credential specification comprises of three type of constraints: a cardinality constraint, a constraint on the issuer, and a condition over its attributes. Cardinality is the number (integer) of distinct credentials (i.e., issued by different entities but of same type) of specified type that subject needs to present in order to be assigned to the specified role. If the cardinality is not specified in a credential specification, its default value is assumed to be one. An issuer is an entity who signs a credential. Issuers can be certification authorities or members of others roles. Roles can also be parameterised. A mapping between the parameters of a role to the attributes of credentials is also specified. In the example below three different roles are defined: `student`, `supervisor`, and `goodStudent`. The role `goodStudent` is parameterized and has two parameters `level` and `recommender`. The parameter `level` is mapped to the attribute `TrustLevel` of credential `trust` and the parameter `recommender` is mapped to the role instance of supervisor or professor who issued the `trust` token. Also note that in following example the employee credential has an attribute named `Role` and should not be confused with the keyword **`Role`**.

```
Role any requires null;
Role student requires employee(Role = "PhD") signed by ICL;

Role supervisor(position) requires employee(Role = "lecturer" OR Role =

"Professor" | position: Role) signed by ICL;

Role goodStudent(level, recommender) requires employee(Role = "PhD")

signed by ICL AND (trust(TrustLevel > 10 | level:TrustLevel,

recommender:x) signed by x:supervisor  OR trust(TrustLevel > 8 |

level:TrustLevel, recommender:x) signed by x:professor);
```

Example 26. Authentication policies

In above example, the symbol "|" is used as a separator. It is preceded by the conditions on token attributes and is followed by mappings of role parameter to token attributes.

4.1.3.2 Condition specification

Before introducing Authorisation and Event-Condition-Action policies we must describe how conditions can be specified over them as this is always critical aspect of policy

expressiveness. Simple conditions can be specified on five types of variables: subject attributes, object attributes, action parameters, values returned by remote and local services. Complex conditions can be built combining simple ones with boolean operators.

Attributes and remote methods can be referenced using the dot (".") notation and previously defined unique identifier of an entity. In particular, object attributes can be referred with name of object variable. A condition over subject attributes can use only those attributes that were listed in the credential specification for the subject (role).

Invocation of methods on remote services must uniquely identify a function or procedure published by the specified remote service. If more than one local service is running, then the invoked one must be as well referenced with a unique identifier. It is assumed that the policy writer knows the published methods and their input and output parameters. The language is not bound to any particular invocation mechanism, but the underlying interpreter must be able to translate the remote service identifier and the method name in an actual remote call.

```
(GSS.time() > 20 AND gpsLocation() != "ICL") OR ICL.isClosed() == TRUE
```

Example 27. Conditions using services

A set of temporal constructs (e.g., within, between, after, before etc.) must be provided to specify conditions over ordering of actions performed. The temporal condition shown in following example evaluates to *true* if the user alice read the specified non disclosure agreement (NDA) within last six months.

within(6 months, historyService.accessTime("alice", "NDA", "read"))

Example 28. Temporal conditions

4.1.3.3 Authorisation Policies

Positive and negative authorisations respectively represent permissions or denials to subject requests for the execution of actions over resources. The keyword **target** is used to describe resources on which the authorization policy is applicable. The object expression consists of an identifier (variable name) and a condition that must be satisfied by target object.

Authorisation policies are constrained by two sets of conditions. We distinguish between initial conditions and monitoring conditions. The former ones are evaluated at the access request time, while the latter are evaluated during the access. If they are evaluated to false, then the access must be interrupted. The two different conditions are specified respectively by the **when** and **while** keywords, as the following syntax shows.

The **with** keyword refers to the list of credentials the subject must possess to be granted access. It helps the policy writer to write clearer and simpler policies avoiding the definition of too different roles. It can also be used to specify consumable credentials, as shown in the example below. The post obligation action markAsUsed is responsible for recording the serial number of the presented security token. If the security token is presented again in future, then it should not be accepted as a valid token.

```
Authorisation studentPermission = Allow read ()
    target o:(o.project="consequence")
```

```
to x:student
  with (t:authToken(action = "read",object = o) signed by
  o.projectManager)
  when (GSS.time() < 20)
  while (location() == x.Office)
  preObligation (GSS.log(x.Surname + "reads" + o.ID))
  postObligation (markAsUsed(t));
```

Example 29. Authorization policy

The example above allows students to read a document if they have received explicit consent from the manager of the project the document belongs to, if the time is less than 20 (before 8 p.m.) and if they are in their offices. Before accessing they must log their request to an external service. After accessing the explicit consent credential must be consumed.

The **preObligation** and **postObligation** key words introduce the sets of actions that must be executed respectively before and after the access execution.

4.1.3.4 Event-Condition-Action Policies

Event-Condition-Action policies represent the obligation to perform specific actions as a response to events occurring in the system. Events carry additional information in the form of parameters that can be used to specify conditions over the validity of the policy. These policies can be used to associate protected data with a behaviour that must be kept after the protected data is received in a remote environment. We assume that the set of behaviours (actions) that can be specified has been agreed in advance between sender and recipient (and thus probably included in the Data Sharing Agreement). Note that these policies are not evaluated as a reaction to an access request.

```
on dateEvent(month, day, year, prevMonth, prevDay, prevYear)
    do delete(o:objectId)
    when (month!=prevMonth);
on receiveMsg(sender,action)
    do delete(o)
    when (sender = o.owner AND action = "delete");
```

Example 30. ECA policy

The elements of policy language presented above provide a lot of flexibility to the policy writer. For example, we have the following options for modelling the authorization rules requiring third party consents:

Option 1: Third party is expected to issue a token to the requester, requester than uses that token to activate roles granting him/her required access. Advantage of this approach is that

design of policy infrastructure does not depend on how the consent token is obtained. However, in this approach the user has the responsibility to obtain the appropriate consent token.

Option 2: Obtaining third party consent is expressed as a pre-obligation in an authorization rule. The challenge in implementing this approach is that the policy and enforcement infrastructure needs to be aware of the process for obtaining third party consent. These processes can be different for each authorization rule. The advantage of this approach is system proactively seeks consent for the user. The authorization process is very simple for the user.

Option 3: Third party consent is seen as an attribute, which can be used as a condition in authorization rule. The disadvantage of this approach is that number of attributes can vary as the policies change. This strategy requires frequent modification to the user profile schema. User is simply denied permission if the consent attribute is not set. Policy infrastructure will not be able to tell the user that he/she could gain access if the user can make that condition true. Neither the policy infrastructure help the user obtain the consent. The advantage of this approach is that it is very simple and functionality of obtaining attribute tokens is already supported by components like STS.

The policy elements discussed above appear to be sufficient for expressing a wide range of policies. However, further work is required for the precise syntax and implementation.

4.1.4 Policy Storage and Policy Deployment

Data sharing agreements are refined into enforceable data sharing policies for all organizations that are signatory to the DSA. These policies are then stored in the policy store of these organizations. A *policy store* provides secure storage for any policies stored in it. All these policies should be available to the policy decision point (PDP) for evaluating access requests. In addition, the policy store should prevent unauthorized modification of the policies stored in it and allow only permitted users (e.g., policy administrators) to read or modify the policies stored in the policy store, or add new policies.

Protected data is usually stored on several locations, such as user's workstations, portable devices, and organizations servers. Some of these locations may not be always online. However, the applicable data sharing policies should be available for the policy infrastructure at the access location. Policy deployment is the process of propagating policies to locations where they may be needed. The policy store used for administrating the policies is considered master policy store. Policies may be deployed at other locations by simply replicating the policy store. However, this approach may not be optimal. We should consider other factors, like the processing power, storage capacity, and need of policies at deployment location.

In situations when data may be accessed from thin client devices or portable devices, and the network availability is not guaranteed, the availability of an external policy store is not guaranteed. Due to resource constraints of device, it may not be feasible to replicate entire policy store, or evaluate the entire policy set (as in policy store) on these devices. However, it may be possible to pre-evaluate the policies, and attach the applicable policies with the protected document. In other words, the attached policies are specific to the document instance. We refer to these policies as *sticky* policies based on the notion that these policies should always stay with the document. Since, the documents can be shared by means like email or copying to a disk, we take the approach of packaging the sticky policy and the protected content in same file. If the sticky policy is stored separately, the data sharing process (e.g., email or copying file) will have to be intercepted, the policy store will have to be queried, and the sticky policy will have to be sent along with the document. All these steps

are not needed in our approach of packaging the sticky policy is same file as the protected content. Note that allowing users to attach policies with documents in presence of data sharing policies can result in conflicts. We defer discussion of such conflicts to Section 4.1.6.4.

In other situations, replication of entire policy store may still be undesirable for performance and security reasons. The set of policies deployed on a device may be significantly reduced based on criteria such as the primary user, and for what purpose that device is used. In many organizations, it is often the case that only a very specific set of users use a given workstation. In our approach, we want to deploy a minimal set of policies that will be sufficient to control access of these users. If a new user logs on to this workstation, additional applicable policies should be deployed dynamically. The set of deployed policies may need to be updated if the user attributes are modified, or if policy set defined at master policy store is modified.

The process of updating deployed policies should be safe in the sense that it should not allow any unauthorized access from the perspective of both old and new policies during the process of deploying an updated policy. Also, it is desirable that if an access request permissible by both the old and new policy set should not be denied during the update process.

4.1.4.1 Policy Decision Point

The primary functions of a policy decision point are to evaluate access requests in a system and to request execution of applicable obligation actions. Due to the complexity of this process the policy decision point needs to be organized into several subcomponents. The core component that evaluates the policies is called the *policy evaluator*. We now discuss the functioning of PDP in detail and role of various subcomponents in it. Figure 3 illustrates all the sub-components of PDP.

Obligation actions may need to be executed as a result of an access, or because an obligation (ECA) rule in the policy needs to be satisfied. Hereafter, we will assume policy evaluation is triggered by receipt of an access evaluation request or an event. We will discuss various types of events in our discussion. A sub-component called *event listener* is needed to receive all events. The job of an event listener is to dispatch events to appropriate event-handling methods in the policy evaluator. Thus, the enforcement layer must know the interface of event-listener. The processing of an event includes evaluation of event based obligation rules and re-evaluation of monitored conditions.

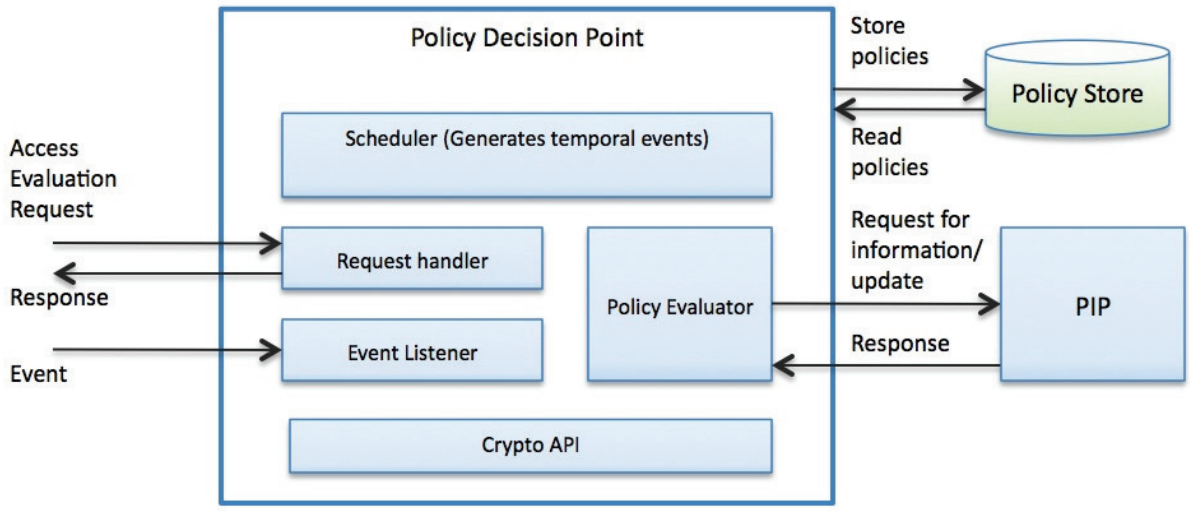

Figure 3. Policy Decision Point (PDP)

Evaluation of an access request may result into obligations that must be executed before allowing the access, during the access, immediately after the access, or at some time in future due to the temporal constraints specified in the authorization rule. The execution of these obligations is responsibility of the enforcement layer. The policy decision point must inform the enforcement layer of these obligations along with the access decision, to allow the enforcement layer to schedule execution of obligations at required timing.

Event-condition-action rules can also specify obligations. These rules may be based on temporal conditions and changes in system state, e.g., receipt of new data, modification of existing data, or change in contextual conditions. The policy evaluator should subscribe to events that are needed to reliable evaluate the deployed policies. For instance, to handle events based on temporal conditions, the policy decision point should have a subcomponent that can generate events based on temporal conditions. We will call this subcomponent a *scheduler*. The policy evaluator must subscribe for required temporal events. The scheduler must then send the generated events to the event listener. While the temporal conditions can be evaluated locally, the PDP is not the best place to determine occurrence of other events. The enforcement layer is the best place to trigger events like receipt of new data and modification of existing data because it intercepts all operations on data in the scope of policy enforcement. Thus, there is a need for communicating about these events between the PDP and PEP, so that the PDP is able to subscribe to events related to change in system state and the environment. Evaluation of contextual parameters like location, environment, or system parameters requires either specialized services (not depending on the policies) or services that depend on the operating system. To keep the implementation of policy decision points independent of operating system dependencies and maintain the focus of PDP only on policy evaluation, these functionalities are abstracted into *contextual services*, located in the policy information point component. The functionality of policy information point is discussed further in Section 4.1.6.

One of the goals of Consequence project is to provide *context-aware* policy enforcement. Contextual information like location, time, and system parameters are best known at the policy enforcement point itself. Usage control policies may require monitoring of contextual information to allow ongoing access. For example, allow Alice to open and read sensitive documents on a mobile device only while the device is inside office building. This policy requires that the location parameter be monitored even after an initial decision of allowing a usage request. If the policy condition evaluates to false, then the ongoing usage must be terminated. This means that such policies (that require monitoring) must be re-evaluated even after grating initial authorization. If the PDP determines that ongoing access needs to be terminated, it should be able to reliably convey this decision to the policy enforcement layer. This means that the PEP must be definitely reachable from the PDP to provide assurance of correct policy enforcement. Only way to provide such assurance is by *collocating* the PDP with the PEP. This also eliminates need for network communication among policy infrastructure components to revaluate decision about an ongoing usage request.

The response of PDP to a policy evaluation request specifies the permissions a user has, and pre-obligation and post-obligations actions that must be executed. Typically in the ERM systems, *a usage license* is created to cache the PDP response along with other information like key used to encrypt the sensitive data. The usage license is assigned an expiration time and date, to ensure policies have to be reevaluated for access over prolonged durations.

4.1.5 Policy Information Point

Evaluation of policies is dependant on the information about subject-attributes, object-attributes, and context. Therefore, first we briefly discuss functionality of Policy Information Point (PIP). The PIP is considered part of the enforcement layer in Consequence architecture and a more detailed description of PIP is given in deliverable document D4.1.

The function of policy information point is to provide information needed to evaluate the policies. When contextual or system information is requested by the PDP, it is recommended that PIP use local services to provide this information. Use of local services eliminates need for network communication among policy infrastructure components to revaluate decision about an ongoing usage request. A PIP may also be used to request information about a subject or object as well. In these cases, the requests need to be forwarded to external services that provide information about subject identity and attributes, or object attributes.

Figure 4 illustrates the various subcomponents of a PIP.

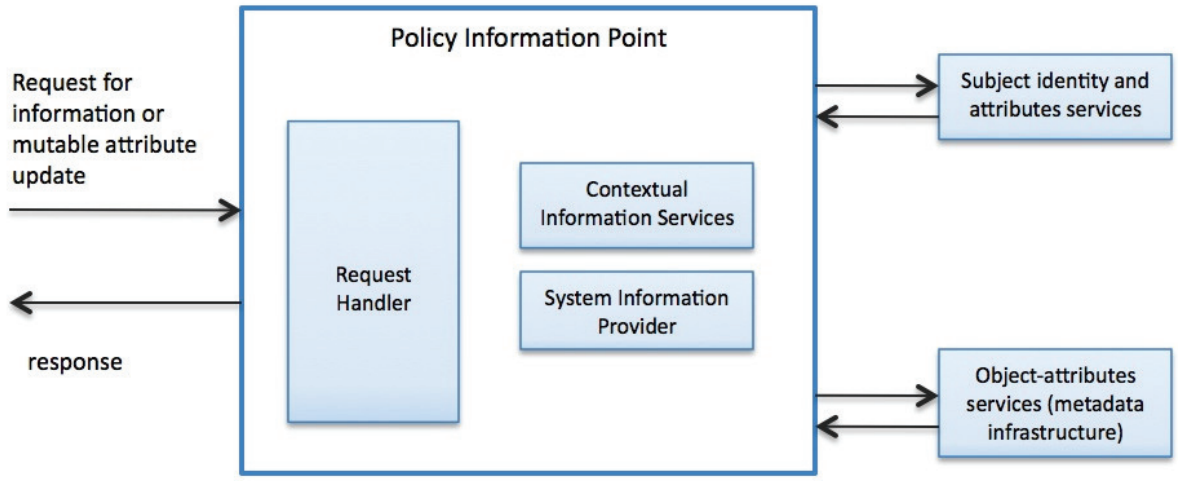

Figure 4. Policy Information Point (PIP)

In Consequence project, the service that provides information about subject identity and attribute information is called the Security Token Service (STS), which is described in D4.1. Intuitively, a STS provides signed attribute information, which may have been obtained from organization's own identity providers (like LDAP or Active Directory) or other federated services. Additional services may be needed to support update of mutable subject attributes. Object attribute services retrieves information from the metadata store, or generates metadata based on business rules of the organization.

We now discuss the process of evaluating data sharing policies, and interaction of policy infrastructure components with other components in the Consequence architecture.

4.1.6 Policy Evaluation and Interaction of PDP with other components

In this section, we will discuss the role of PDP in more detail and specify its interactions with other components of policy infrastructure. We discuss some functionalities of the Policy Enforcement Point (PEP) for completeness of our discussion. A complete and more detailed description of the PEP is given in deliverable document D4.1.

The scenarios for interaction with the PDP in Consequence architecture can be divided into three categories: sending data, receiving data, and controlling access to shared data. We now describe these scenarios explaining interaction between components of policy infrastructure.

Without loss of generality, consider two organizations that are signatory of a data sharing agreement. Both these organizations may act as data providers and data consumers. Also, under the data sharing agreement, an organization may be required to send data to other organizations when a certain event occurs. For example, in the crisis management test bed, when a gas utility company is informed of a fire incident, it should allow control room officers to critical gas pipe routes, pressure distributions over an area, etc. in the vicinity of the incident. The gas company may choose to provide this information by sending a message (Tactical Situation Object). To enforce this kind of requirement, it is necessary that obligations specified in data-sharing policy be evaluated and enforced even if an organization is acting only as a data provider. This means that policy decision points and enforcement point must be located in organizations that may act only as a data provider. Note that in order to prevent unintended exposure/sharing of sensitive data, either all outgoing communication channels should be controlled, or the data should always be cryptographically protected such that an unauthorized dissemination of document does not compromise security of the protected data.

4.1.6.1 Sending Data

In an implementation of the data sharing agreement, an event or a user request may trigger the process of sending certain protected data to a recipient. We now consider examples for each of these cases and discuss required interactions for them.

- *Temporal event as a trigger for data sharing*: Temporal events play an important role in fulfilling data sharing requirements that cater to preserving integrity and availability of data. For instance, classification of data may change when certain time interval elapses and data must be disseminated to external entities for making it available to other data consumers. In other cases, providing updated data at regular intervals is a trade-off between efficiency and integrity acceptable in some applications. Policies based on temporal events may also be used to fulfill data sharing obligations of an organization. To illustrate processing of such policies, let us consider a data sharing policy that requires STFC to send data collected during a month from its beamline experiment to another organization on last day of that month. The rule is applicable in STFC's domain, therefore the PDPs schedule an event to trigger data dissemination on last day of each month. When the event is triggered, it is directly sent to the event handler of the PDP. The policy evaluator may need to evaluate other conditions specified in the event condition action rule. To evaluate these conditions, the policy evaluator needs to communicate with the policy information point. However, in this example there are no other conditions. Therefore, the policy evaluator simply forwards its decision to execute the obligations to the response builder component, which in turn sends it to the collocated PEP. The collocated PEP must then execute the obligation of sending data as specified in the message from PDP.
- *Receipt of data as a trigger for data sharing:* Consider the scenario of crisis management, where the fire department has a data sharing agreement with the local gas company. The agreement says that in event of a fire, the fire department is required to inform the local gas company about the incident. When fire department receives the information about a fire incident, it will trigger the organization's processes and in particular trigger the process of sending data about incident to the gas company. From an architectural perspective, there are two options: the event invocator could be built into the fire department's application, or the event invocator could be built into the underlying database. Adding this functionality in a database product will eliminate need for custom coding and only configuration of events will be needed.

Alternatively, in some organizations presence of certain files in the file system can be used to trigger events. Due to application dependant nature of event invocators, enforcement layer is an appropriate place for their implementation. The events should be sent to event listener in the PDP, which can then evaluate conditions in related ECA rules and determine applicable obligation actions, which include data dissemination actions.

- *User initiated request for data sharing*: When a user submits a request to send data, the process to check permission for this operation is same as the process (discussed later in this section) of checking user's permission for any other type of access. The process of controlling access to a resource is discussed later in this section. In cases, where the outgoing communication channels are not controlled, the sensitive data is needs to be protected using encryption, such that unauthorized users may not decrypt the data.

Figure 5 illustrates the interactions among policy infrastructure components to check whether an operation to send data is permitted or not.

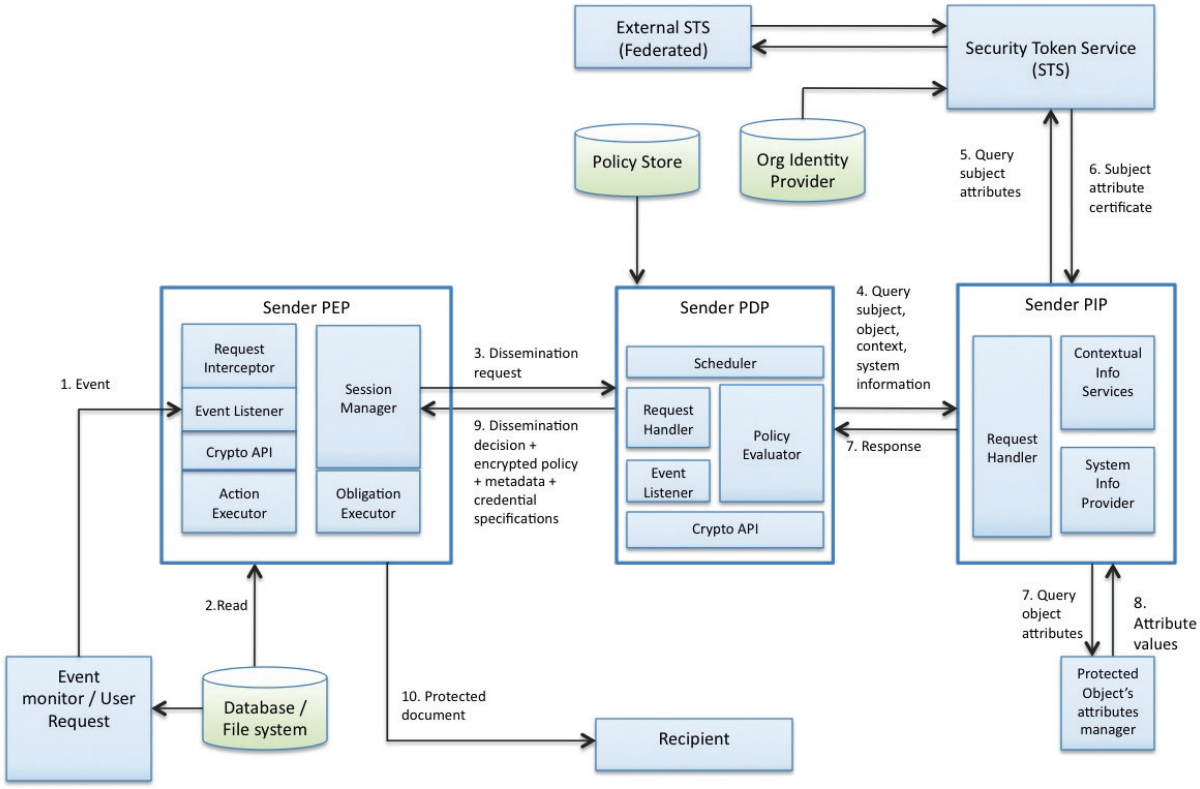

Figure 5. Sending data.

After evaluating the request to send data, the PDP responds to PEP with its decision. The PDP response to PEP comprises of any additional metadata that should be attached with protected data, sticky policy (which may be encrypted if it is confidential), list of trusted certification authorities, and a list of obligation and conditions that must be satisfied if the dissemination was allowed. The PDP may specify certain obligation actions even if the dissemination request was not allowed. In addition, the recipient must be aware of credentials that he/she must provide to gain access to protected data. Therefore, a list of credential specifications must be attached with protected data. A credential specification describes security tokens (attribute name, type, acceptable certifying authority) the user must present with their request to access a resource. Thus, the policy decision point should also include credential specifications, in its response to the enforcement layer.

In the above discussion, we have discussed some implementation specific steps that need to be performed when sending protected documents. Additional policy-based behaviour may be desired on receiving data, which we discuss next.

4.1.6.2 Receiving Data

On receiving protected information from a partner, the recipient may be obliged to act on it. Distinguishing the stage of receiving data from the stage when a user accesses data is useful, especially when data sharing is done automatically, i.e., the sender and receiver are software applications in reality. For example, when a fire department informs a Gas company about a fire near its gas pipeline. The data sharing agreement between them may require the gas company to send alerts to its workers requesting to shut-off gas supply in effected pipeline. In this scenario, an application in fire department may invoke a web service in the gas company and submit details of a fire near its gas pipeline. The web service then stores the received document in a container for protected content (PEP). At this stage, policy to execute obligation actions must be enforced.

The above scenario is very similar to the case of sending data where received data is used as a trigger to initiate request for sending data. However, the scenario is different because 1) the obligation that needs to be executed on receipt of data may not be always to send data, and 2) confidential sticky policies attached with the received document need to be processed. Other kind of obligation actions may need to be performed at this stage. Figure 6 illustrates the sequence of interactions among components of policy infrastructure on receiving protected data.

On receiving new data the event invocator located in the enforcement layer sends an event to the PDP. The event description should comprise of metadata received with protected data, sticky policies (if any) attached with received document, any credential specifications provided. If the attached sticky policy is encrypted, the event description must also include list of PDPs (should be provided with protected content) trusted to evaluate the encrypted sticky policy. The recipient-side PDP must communicate with sender specified trusted PDP to evaluate the confidential policies. Confidential policies may be pre-evaluated when they are received. Pre-evaluation of the confidential policy will eliminate the need for remote evaluation of confidential policy. This will provide a faster response time when a usage request is made. We discuss evaluation of confidential policies further in Section 4.1.6.4. The recipient PDP will store the results of confidential policy evaluation in its policy store, which will be used later to check authorization of usage requests.

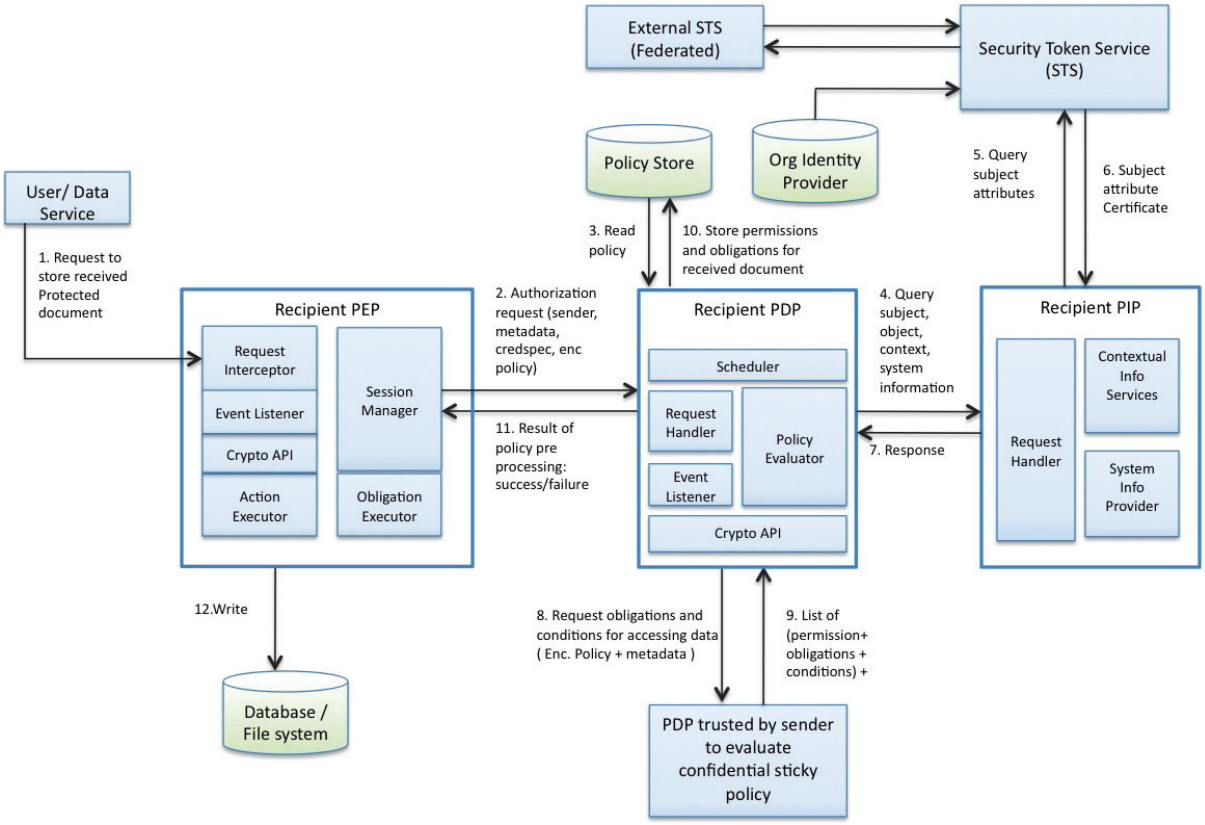

Figure 6. Receiving Data

4.1.6.3 Controlling Access

Controlling access of users is a concern for organizations irrespective of whether they are participating in data sharing agreements or not. In Figure 7, we show the scenario when access request is made in a recipient organization. In our discussion we will also consider the case where access needs to be controlled according to an organization's security policy also. The policy evaluation process for these two scenarios is very similar, therefore we keep our following discussion general and we will explicitly mention differences in them.

When a user accesses protected data, the PEP is responsible for intercepting the usage request. The PEP must then communicate with the collocated PDP to check whether the user is authorized to perform the requested access. An important characteristic of *usage control* systems is that they should be able to control any ongoing access operations. This makes it necessary for the enforcement layer and the policy evaluation engine to distinguish between authorization checks for new access requests and those for ongoing access operations. In other words, the PEP and PDP must incorporate a notion of access session in their architecture.

When a PDP is asked to check authorization for a new access request, the policy evaluator evaluates applicability of all policies in the policy store. This is determined by evaluating object expression of each rule. The object expression specifies conditions over the attributes of protected content (metadata), if these conditions are satisfied a policy rule is considered applicable. The applicable rules may originate from the sticky policy, a data sharing policy, or an organization policy. If the data sharing policy or organization policy has changed after protecting the document, it is expected that the new policies are already stored in the policy store. However, in case of sticky policies the only way to determine if they are still valid is to check with the data provider. Thus, at this stage communicating with data provider's PDP is necessary to be able to implement revocation of rights granted using sticky policies.

Next, the policy evaluator determines the roles that may satisfy these policy rules. Given the definition of roles and credentials in the policies, the policy evaluator queries the policy information point in order to verify whether the user requesting access has these credentials. The PIP dispatches this request to the security token service. The security token service is responsible to respond with a certificate specifying attributes that a user has and which certifying authority provided the attribute value. In the STFC test bed scenario, note that the users of external organizations are also allowed to use the IT resources of STFC. Thus, when the user attributes are requested during policy evaluation, the STS may need to query the security token service of external organization to obtain that information.

After receiving information about subject attributes, the policy evaluator can now determine the roles that current user can acquire. Next step is to evaluate the conditions specified in the policy rules. The policy evaluator then returns a decision to the enforcement point. In addition to the authorization decision, the response of PDP may also specify the applicable pre-obligations, and post-obligations that must be enforced. Moreover, if the authorization was provided by policy rules that specify security requirement for ongoing access, then the PDP response to PEP must indicate this fact.

To evaluate the permissions for ongoing access, the PDP needs to evaluate only the monitored conditions specified in policies that provided authorization. If control over ongoing access is needed, the policy enforcement point must be able to terminate ongoing access when the PDP says so. Authorization check for ongoing access can be performed in two ways: push and poll mode. In poll mode, the enforcement layer polls the PDP for authorization decision after regular intervals during the lifetime of access session. Where as in the push mode, the PDP pushes any change in authorization to the PEP in the form of an event, and the PEP informs the PDP when an access session terminates. The same design options exist for interaction between PDP and contextual services for evaluation monitored contextual conditions. Polling is an expensive activity in terms of resource consumption. It is best to use push model, where the responsibility to send events is assigned to source of information, e.g., in case of contextual services, the contextual service should generate an event and inform the PDP. The PDP must reevaluate affected policies and send an event to PEP describing its new decision.

As part of executing obligation actions, the PEP may be in effect updating mutable attributes of the user. This functionality should be abstracted as a separate component and must be managed by the policy information point, as shown in Figure 7.

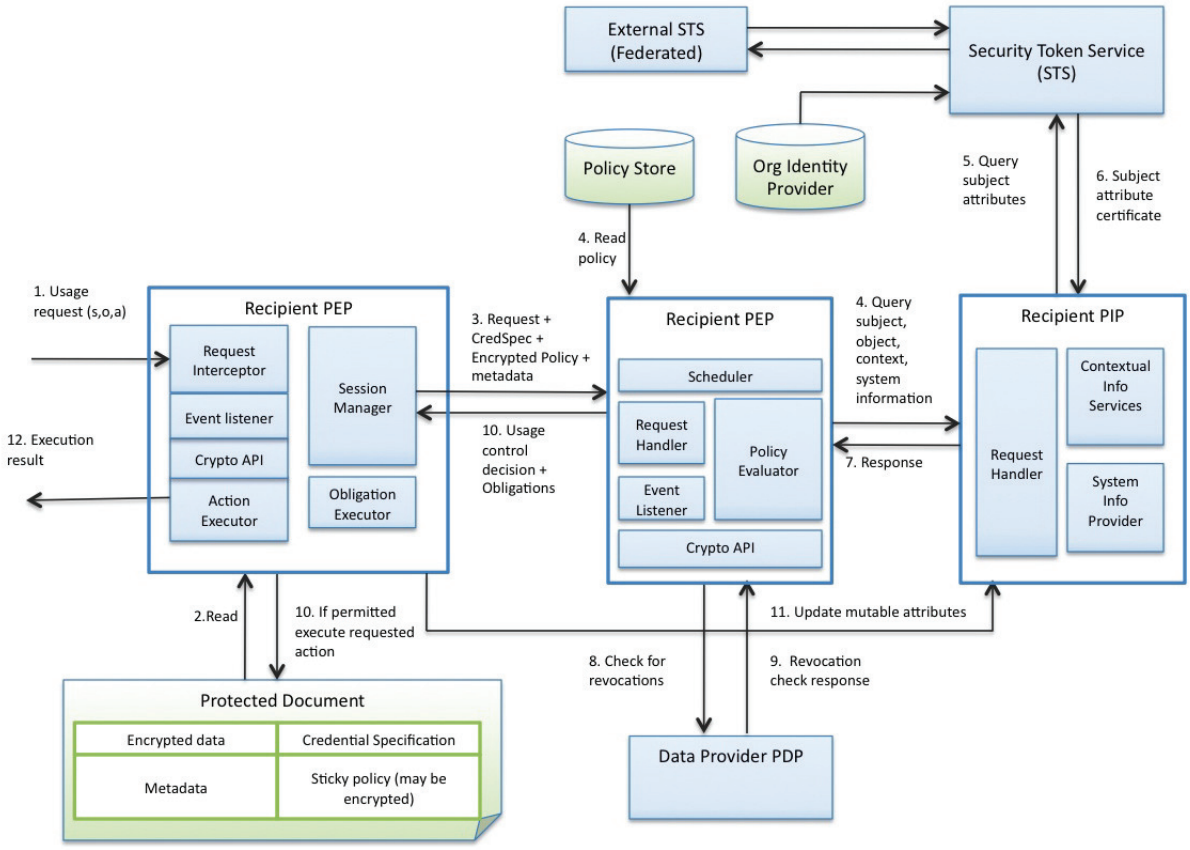

Figure 7. Controlling access to protected document

4.1.6.4 Evaluation of Document-specific Policies

In see Section 4.1.4, we recognized reasons for which document specific or sticky policies may be used in data sharing. These include ease of policy deployment on devices with limited resource, and to keep any unique security requirements (like patient specified policy for medical records) together with data to which they apply. The mechanisms for evaluating them differ based on the reason for which they are used. It may be noted that if the purpose of document-specific policy is ease of deployment, the policy to be attached can be determined by the policy infrastructure itself. However, in second case, a user is specifying the attached policy.

When a user manually requests data sharing, the user may specify a sticky policy to be attached with the document. When a document with a sticky policy needs to be sent from one organization to another domain, it raises new concerns, like: 1) the policy specified by a user should not be conflict with the DSA or the organizational policy, 2) how to define and control the scope of sticky policy rules that a user may specify, and 3) which authorities can be mutually trusted by sender and receiver to evaluate the sticky policy. These are complex issues and we are researching such questions. We now discuss these problems in more details and present some examples to illustrate the meaning of above-mentioned problems.

For a sticky policy to be meaningful, it is necessary that they are satisfied before granting a user access to the content protected by it. However, a user should not be able to use sticky policy as a tool to violate the data sharing agreement. This means that the sticky policy should not be able to deny permissions needed to perform obligation actions as required by the data

sharing agreement. For example, if a DSA says that organization X *must* share its reports of project P with all members of project P in organization Y, the author (belonging to organization X) of a report on project P should not be able to add a sticky policy denying access to all members of project P in organization Y. However, if the DSA says that employees of organization X will be allowed to share their reports of project P with members of project P in organization Y. Then, an author (belonging to organization X) may be allowed to attach a sticky policy denying access to members of project P in organization Y. This example shows how conflicts may occur between a sticky policy and data sharing policy, and why it is important to control the scope of sticky policies that a user may specify.

Let us now discuss the problem of using a mutually trusted authority for policy evaluation. Policies are specified in terms of object attributes and on the set of credentials users present. From the perspective of recipient, policy evaluation can be considered safe when the evaluation authority and the recipient are part of the same organisation. In the more general case addressed by ERM systems, multiple organisations cooperate and exchange data. This situation leads to the following two shortcomings. First of all, policy writers must be aware of the partner organisation's internal structures. They must know internal roles, hierarchies, and groups to write sufficiently detailed policies to effectively mitigate data misuse. The second shortcoming is related to preserving confidentiality of the users' credentials. Existing protocols assume that users requesting access to data are willing to issue their credentials to an unknown verifier entity designated by the originator organisation. In real scenarios, this could lead to users' privacy concerns. Furthermore, this create a completely unbalanced situation where the users being evaluated have no influence at all on the evaluation process and can not decide which entity will be the verifier.

A similar issue has been addressed in the research field of trust-management systems. The frameworks and languages developed in this field address the problem of distributing trust and more generally policy evaluations over different (possibly unknown) entities. Examples are PolicyMaker (Blaze, Feigenbaum, & Lacy, 1996), REFEREE (Chu, Feigenbaum, LaMacchia, Resnick, & Strauss, 1997), KeyNote (Blaze, Feigenbaum, & Keromytis, 1999), SPKI/SDSI (Clarke, Elien, Ellison, Fredette, Morcos, & Rivest, 2001), the Trust Policy Language (Herzberg, Mass, Mihaeli, Naor, & Ravid, 2000) and the RT framework(Li, Mitchell, & Winsborough, 2002). For the sake of simplicity in the following we will refer to the SecPAL language (Becker, Fournet, & Gordon, 2005) as the most representative of the whole.

The core idea is that a policy can be evaluated on the base of the results of several delegated decision processes. In other words the right of evaluating a specific condition can be passed from entity to entity. The result is the construction of a complex chain of delegated evaluations that the policy writer trusts. The SecPAL language allows the definition of several constraints to drive the chain construction such as conditions on the delegating entity attributes, on the context, on the length of the chain etc.

Application of this solution to cross-domain data sharing systems would bring several advantages. It would allow to specify sets of policy evaluators among which a user being evaluated could choose and to write policies without knowing the internal structure of partner’s organisations. There are however also several shortcomings. First of all, what we called *policy evaluation* in trust-management frameworks is actually a simple attribute assignment. What is possible to delegate is the decision over the assignment of one or more attributes to an entity. Delegated entities do not actually evaluate a policy issued by the policy writer but simply returns a credential representing the association of a user to an attribute in their local namespace. The policy writer has no control over the criteria that has been used to

make these assignments. This lack of control is an important issue especially when high-value resources are disseminated.

We must also consider the problem of actually evaluating the defined policies. Existing solutions are based on the representation of each delegation step and attribute assignment with signed credentials. To evaluate a policy all the credentials that can build a valid chain must be gathered and passed to a central evaluator. ERM systems are usually based on centralised credential stores. Considering this implementation too simplistic, the approaches proposed by trust-management frameworks usually opt for making either the resource owner or the access requester gather them from the various entities in the system. Both the solutions have anyway some problems. First of all it is necessary to assume that there exists a link between the credentials searcher and all the nodes in the chain. This implies that all entities know identities and locations of the other ones. Second, each entity in the chain must be willing to issue its own delegation and trust policies, so that the credential searcher can know what node in the chain it must ask for credentials at the next step. Finally, all the credentials must be passed to the evaluator, expanding the privacy issue to all the entities in the chain.

The core idea of our solution is to couple policies so that for each policy, the set of possible evaluators is restricted by another policy. This leads to the creation of a *chain of policies* where each step (or level) represents the delegation of the right to evaluate the previous policy. This also means that users (or evaluator entities) that must be evaluated against a policy might choose their trusted evaluator among those satisfying the higher level one. Assuming policy writers agree on specify such policies, privacy protection and the evaluation process unbalance should not be an issue any more.

The problem of trust transitivity and loss of control over a resource is also addressed. The form and growth of the delegation chain is in fact completely controlled by the policy chain. Policy writers can specify exactly what different requirements entities must satisfy to be part of each evaluation chain level. Figure 8 depicts a policy chain construction. Each policy defines a set of entities that satisfy it and can evaluate the underlying one.

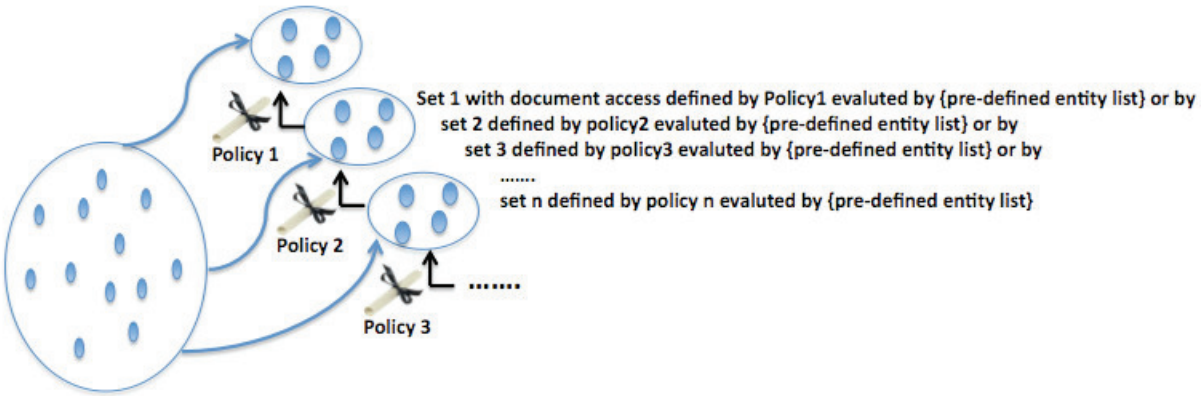

Figure 8. Policy-defined groups

The term *policy* is used here to identify an authorisation rule containing not only attribute verifications but also provision requirements (i.e. action that must be executed before the authorisation has been granted) and context-dependent conditions. We intend to develop a policy language (or a modified version) similar to the SecPAL language that can be used to build up a policy chain.

The distributed evaluation process can be then made possible by hiding inside the disseminated resource all the information needed to access it (encryption keys and policies). For each arc of the policy chain the two involved nodes perform part of a recursive protocol unlocking part of the hidden information. Each unlocked piece of information allows the execution of the next iteration of the protocol. Only when all the iterations have been completed and all the information unlocked, the resource can be accessed. This process ensures that the document can be accessed only if a trusted evaluation chain (where each step returns a positive evaluation) exists and allows distributing the effort of its evaluation between all the involved entities.

4.1.7 Discussion

In this section, we have described some key decisions made in the proposed architecture and justifications for them. In summary, we have proposed to always keep protected shared data encrypted, add metadata to describe protected data, and allow instance-specific policies to be attached with protected document. We have also described the interaction between different components of policy infrastructure under different scenarios.

The scenarios described earlier in this section, will drive the interface that will be provided by an implementation of PDP. For example, some of the questions the PDP should be capable of answering are:

1. Is a usage request by a subject permitted, given requested permission, credential specification, encrypted policy, and object metadata?
2. Is a usage request by a subject permitted, given requested permission, credential specification, policy, and object metadata?
3. What is the list of permissions and associated obligations and conditions for a given object that allow a given subject to access it?
4. Given a dissemination request and recipient, what are the applicable policies for an object?
5. Given an object and policy, determine the metadata that is required to evaluate the policy?

A recipient of a protected document may want to attach additional metadata to a received document and redistribute the document when permitted.

4.2 Metadata Generation Infrastructure

In this section, we describe the different operations of the metadata infrastructure during the *Bootstrap*, *Dissemination*, *Receiving* and *Usage* phases. This infrastructure consists of a Metadata Management System (MMS), which draws its taxonomy from a Data Description Ontology (DDO) and a Data Security Ontology (DSO).

4.2.1 Bootstrap Phase

The *Bootstrap Phase* describes how initial or derived data, which has no metadata information, will be first introduced into the metadata infrastructure.

Typically, in any computing system, data may be *generated* through some application, *acquired* from other systems or the context, or *derived* from other data using some analysis

tool. We assume that this creation of new data happens through some Data Generation/Acquisition/Derivation (DGAD) system(s). For example, in scientific experiments, special sensors may be positioned in the physical environment where the experiment is taking place such that these sensors will convey information about the experiment in the form of raw data that is captured by a data acquisition system. Later, this raw scientific data may be analysed and new derived data will emerge.

Once the data has been generated/acquired/derived by the DGAD system, it will be stored in a Data Storage system for later access by other systems. In effect, the DGAD system will bootstrap the Sender's metadata and policy infrastructures with the initial raw data the Sender is going to disseminate in the future.

Once the data has been stored, the DGAD informs the MMS that it should mark-up the new data with its descriptive metadata, drawn from an ontology related to the domain of the data. For example, if the data is generated by scientific experiments, then the Metadata Management System could be a cataloguing service such as the ICAT system (http://www.metataxa.com/icat.asp), which itself would be based on some scientific ontology such as the CCLRC Metadata Model (CCMD) (Sufi, S., Matthews, B., Kleese van Dam, K., 2003, Sufi, S., Matthews, 2004), which describes the domain of scientific studies and experiments.

In case the data is derived from other annotated data, then the MMS will derive metadata from the new data.

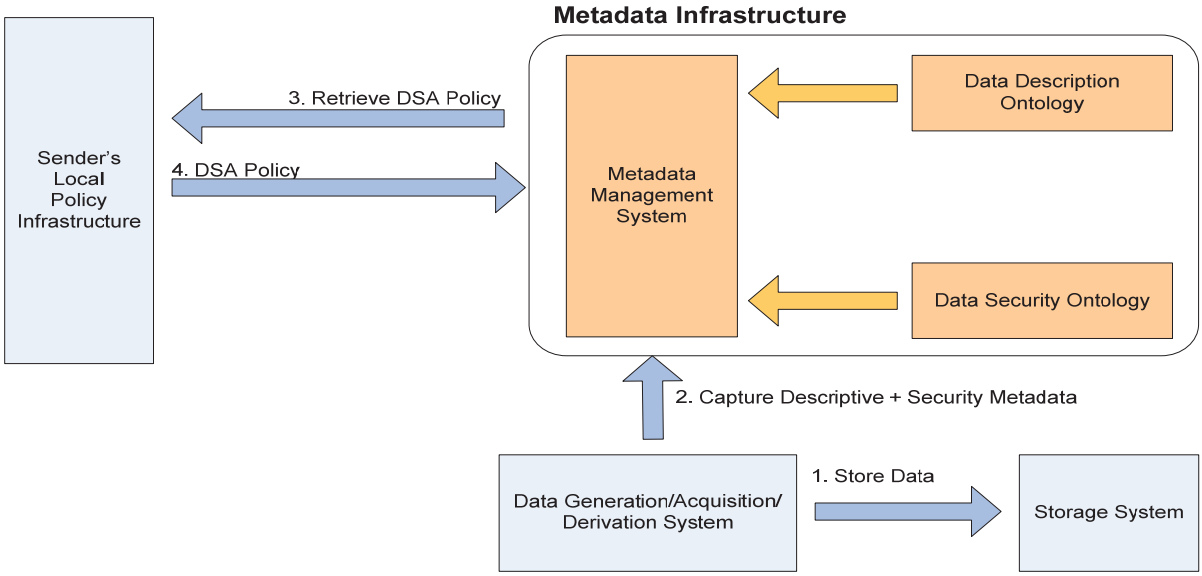

Figure 8. Metadata Infrastructure during the Bootstrap Phase

In Consequence, the general descriptive metadata associated to some data could be necessary for evaluating security policies, as it conveys some of the attributes of objects (data) in access and usage control policies. However, these descriptive metadata may not always be sufficient for evaluating security policies. For example, if the security policy for accessing the data mentions security classification levels, such as *Public*, *Confidential* and *Top-Secret*, the corresponding data will need to carry similar information, i.e. will need to be classified at one such level. Therefore extra security-specific information may need to be included in addition to the general description of the data. This raises the need for a Data Security ontology,

which will be based on the terminology of the DSA the Sender shares with the Receiver. Later in Section 4.2.6, we introduce a general Data Security ontology based on OWL.

In the next step, the metadata infrastructure will contact the Sender's policy infrastructure in order to retrieve the DSA policy controlling the dissemination of the new data. Once this is obtained, the MMS will then mark-up the new data with the necessary security metadata.

4.2.2 Dissemination Phase

During the *Dissemination Phase*, the Sender receives a request or an event occurs, which requires it to send the data to some destination. The interactions between the policy infrastructure and the metadata infrastructure are shown in Figure 9 below.

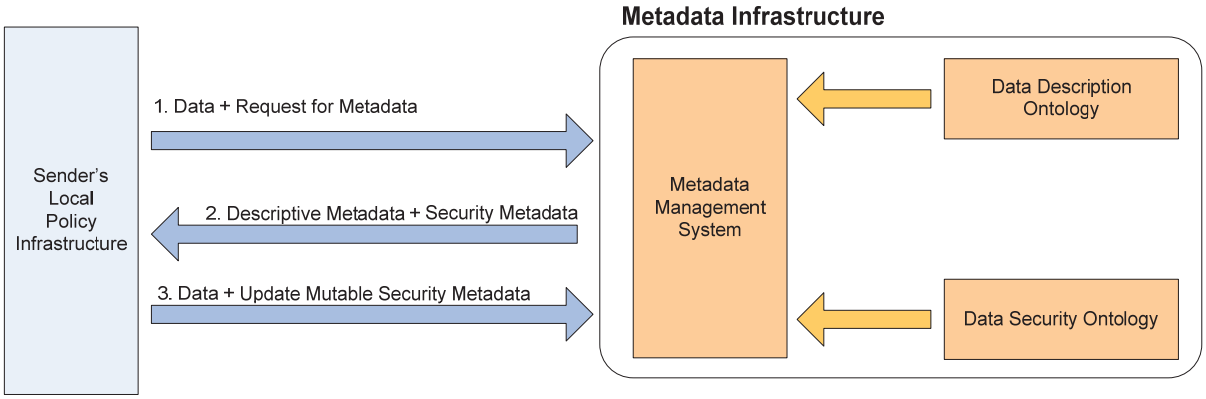

Figure 9. Metadata Infrastructure during the Dissemination Phase

Note that we assume that only security metadata may be mutable, i.e. all descriptive metadata is immutable. The interaction starts when the Sender's policy infrastructure requests from the MMS the metadata corresponding to the data being disseminated. The MMS replies with the relevant descriptive and security metadata corresponding to the data. Finally, the Sender's policy infrastructure may send an update of the mutable security metadata, such as any log information on the data request, or even modifying the security level of the data. This could be necessary for example if the embargo on certain confidential data has expired and it is now possible to disseminate it to the general public. This will degrade its classification level to Public.

4.2.3 Receiving Phase

In the *Receiving Phase*, the recipient of some data along with its metadata informs the local metadata infrastructure of the new data. The interactions are shown in Figure 10 below.

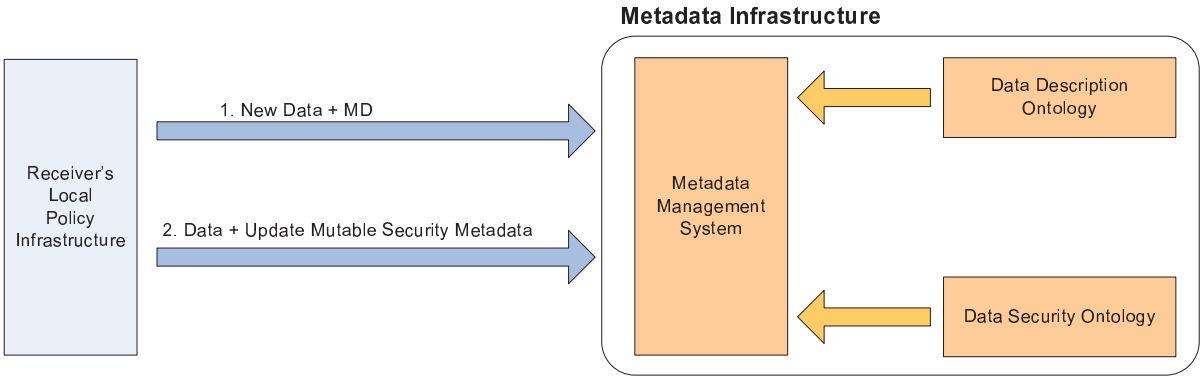

Figure 10. Metadata Infrastructure during the Receiving Phase

Note that this phase is different from the Bootstrap phase in the sense that the latter may have to deal with data that have no metadata, whereas here, the Receiver always receives annotated data. Finally, the Receiver's policy infrastructure may also update any mutable metadata of the received data.

4.2.4 Usage Phase

In the *Usage Phase*, the User's local policy infrastructure queries the metadata infrastructure about the metadata of the data being used. This interaction is shown in Figure 11 below.

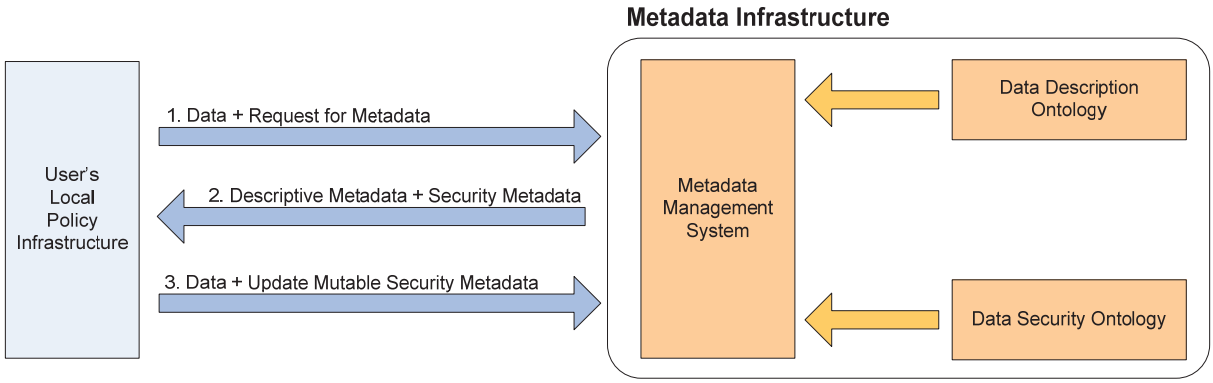

Figure 11. Metadata Infrastructure during the Usage Phase

The User will request the metadata from the MMS after which the MMS replies with the descriptive and security metadata. Finally, the User may update mutable security metadata.

4.2.5 A Taxonomy for Data Description

There are several ontologies in the literature on security terminology (Anya Kim, Jim Luo, Myong H. Kang, 2005, Grit Denker, Lalana Kagal, Timothy W. Finin, Massimo Paolucci, Katia P. Sycara, 2003). However, none of these deals with data security in a specific manner. Therefore, we introduce in this section, a general taxonomy (class hierarchy) of data security based on the OWL language. This taxonomy is shown in Figure 12. The taxonomy is motivated by the metadata requirements introduced in Section 2.1.2.

4.2.6 A Taxonomy for Data Security

There are several ontologies in the literature on security terminology (Anya Kim, Jim Luo, Myong H. Kang, 2005, Grit Denker, Lalana Kagal, Timothy W. Finin, Massimo Paolucci, Katia P. Sycara, 2003). However, none of these deals with data security in a specific manner. Therefore, we introduce in this section, a taxonomy (class hierarchy) of data security based on the OWL language. This taxonomy is shown in Figure 12.

This taxonomy introduces a new class called **DataSecurityInformation** , which is a subclass of the top class, **Thing**, and which consists of the following subclasses:

- **Redundancy Information**: This class includes any redundancy information necessary for checking the integrity of the data. It consists further of the following subclass:
    - **Checksums**: this subclass represents the classical error correcting codes, such as the *cyclic redundancy check*. Notice that cryptography hash functions are subclasses of the Cryptography Information class discusses below.

- **Freshness Information**: This class defines information necessary to reason on the freshness of data. It consists of the following two subclasses:
  - **Timestamp**: usually contains the time and date at which the data was created.
  - **Nonce**: This is a random number, which must be fresh each time new copy of the data is produced.
- **Contextual Information**: This is information on the permissible contexts in which the data may be accessed. The following subclasses represent some of this information:
  - **Permissible Time Range**: This information indicates the time range in the day during which the data may be accessed/used.
  - **Permissible Date Range**: This information indicates the date range in the year during which the data may be used/accessed.
  - **Permissible Location Set**: This information represents the set of locations in which the data is allowed to be accessed or used.
- **Classification Information**: This class contains two subclasses that are used to classify the data:
  - **Security Level**: This subclass represents the security classification of the data. This classification could be taken from either a total ordering relation (e.g. Public < Confidential < Top-Secret) or from a complete lattice structure (e.g. the powerset lattice).
  - **Conflict Class**: This subclass represents the conflict group to which the data may belong. Conflict groups are necessary to avoid accessing/using data belonging to the same group at the same time, which could lead to breach of security properties, such as confidentiality or anonymity.
- **Provenance Information**: This class represents the usual provenance information, which at a minimum includes the following subclasses of information:
  - **Place of Origin**: This subclass represents the original place (e.g. IP-domain) in which the data was first created.
  - **Data Ownership**: This describes who the owner of the data is.
  - **Data Lineage**: This subclass represents the original time and date at which the data was created.
  - **Original Identity**: This subclass represents the original identity of the data (e.g. unique identifier).
- **Cryptography Information**: This class contains any cryptography information associated with the data. This could be any of the following subclasses:
  - **Digital Signatures**: This subclass includes any digital signature, such as DSA, RSA and El Gamal signatures.
  - **Hash Functions**: This subclass represents cryptographic hash functions, such as SHA-1, MD-4 and MD-5.
- **Environmental Information**: This class represents information on any environmental constraints on the access, usage and routing of the data. Currently, we envisage the following two subclasses for this class:
  - **Permissible Network Route**: This subclass represents the permissible network routes that the data may be communicated over.
  - **Permissible IP Domain**: This subclass represents the permissible IP domains in which the data may be accessed and used.

Figure 12. A Taxonomy for Data Security.

- **Safety Information**: This subclass contains any safety information on accessing, using or routing the data. Currently, it contains a single subclass:
  - **Permissible Operations**: This subclass contains the list of operations that can be applied to the data. This could be either usage operations, such as read or write, or communication operations, such as send.
- **History Information**: This class contains information about the history of the data. It consists of the following subclass:
  - **Logs**: This subclass represents the usual log files.

4.3 Analysis against requirements

4.3.1 Analysis of Policy Infrastructure requirements

We now discuss how the proposed architecture and policy language satisfied the requirements. The proposed policy language and architecture described earlier in this document satisfies many of the basic/high-priority requirements. Our proposal also satisfies some of the advanced requirements. A summary of progress is illustrated in Table 3 below.

Requirement		Addressed (Partially = Work in progress, TBD = To be done)
PDR 1.	Attribute based description	Yes
PDR 2.	Trust relation for obtaining attribute values	Yes
PDR 3.	Trust relation for evaluating security policy	Partially
PDR 4.	Authorization based on third-party approval	Yes
PDR 5.	Threshold based authorization	Yes
PDR 6.	Support for controlling manual dissemination	Yes
PDR 7.	Access based obligations	Yes
PDR 8.	Event condition based obligations	Yes
PDR 9.	Evaluation of conditions	Yes
PDR 10.	Deterministic evaluation	Partially
PDR 11.	Partially offline evaluation	Partially
PDR 12.	Threat mitigation	Partially
PDR 13.	Non repudiation	TBD
PDR 14.	Protection of derived data	TBD
PDR 15.	Protection of document parts	TBD
PDR 16.	Data transformations	TBD
PDR 17.	Delegation	Partially
PDR 18.	Policy references	TBD
PDR 19.	Privacy protection	Partially
PDR 20.	Sticky policies	Partially
PDR 21.	Policy modification	TBD

Table 3. Policy infra requirements - progress summary

To make policy infrastructures interoperable across domains, we have suggested two-pronged strategy. First, we have suggested use of metadata to describe object attributes and subject attributes. The proposed policy language allows authorization of subjects based on their credentials, where each credential can be an assertion about one or more user attributes. Also, the protection objects are identified based on metadata associated with them. However, our

model assumes that the protection requirements are data-independent. These contributions satisfy requirement PDR 1.

The policy language also allows the policy writer to specify authorities trusted to issue any security token presented to the policy infrastructure. This is achieved through use of **signedby** clause in the authorization policies. A security token can be used represent a certified attribute value, or an approval from third party. Thus, allowing us to satisfy requirements PDR 2 and PDR 4. The authorization policies also allow expression of a minimum number (threshold) of security tokens that must be presented by a user to gain access to a resource. This feature helps us in fulfilling requirement PDR 5.

In order to model and implement trust relationships with external entities for evaluation of security policies, we have introduced the notion of policy chains and the protocol for implementing such policy chains is under developed. Thus, we hope to satisfy requirement PDR 3 in near future.

The proposed model allows the policy writers to define permissions for manually dissemination data in same way as authorization policies are written for other access types. Also, the sensitive data is stored in an encrypted form to prevent an adversary to bypass legitimate dissemination channels. Together, these features allow support for controlling manual dissemination of data (PDR 6).

The authorization policies described in proposed policy language allow specification of obligation actions. Obligations that should be executed before access is authorized are called **preObligations**, while obligations that should be executed after usage request completes are called **postObligations**. Obligations can also be expressed as event-condition-action rules. The policy infrastructure informs the enforcement layer about the obligation actions that need to be executed. This allows us to satisfy requirements PDR 7 and PDR 8.

The policy language allows us to express pre-requisite conditions for authorizations as well as conditions that should be satisfied during ongoing usage of protected resource. The pre-requisite conditions are specified using **when** clause, and the ongoing conditions are specified using **while** clause in an authorization rule. The architecture also supports evaluation of policy or partial policy at PDP collocated with the enforcement point. This allows evaluation of contextual conditions. The language also supports user-defined functions to access domain specific parameters required to evaluate the conditions. Thus, satisfying requirement PDR 9.

It is often the case that several security policies are applicable to a given request. We have mentioned different types of conflicts that can arise among these policies. Typically, the analysis techniques provided in the DSA architecture provide for the detection of these conflicts however in advanced scenarios conflicts may arise at run-time and cannot be detected and resolved statically. In these circumstances it is necessary to resolve the conflicts at run-time, typically by prioritising the policies to achieve deterministic evaluation (PDR 10). We have starting working on this problem but it will need to be periodically revisited throughout the project whenever the policy language expressiveness evolves.

To support partially offline evaluation of policies (PDR 11), the policy infrastructure supports features like document-specific policies and delegation of policy evaluation authority. At present we are working on detailed specification for implementing these features and hope to satisfy this requirement next year. Our current work on document-specific policies and delegation of policies also indicates that we are making progress towards addressing requirements PDR 17, PDR 19, PDR 20. We hope to address these issues in next year of the project.

Current architecture of policy infrastructure address many threats (PDR 12) by requiring signed security tokens for subject and attribute information, controlling access to policy store, and using policy-based trust relationships. However, more work is needed to address complex issues like controlling abuse of privileges given to users, and controlling the scope of policy-writing privileges given to individual users. Time permitting; we hope to address such complex issues.

4.3.2 Analysis of metadata generation requirements

The metadata infrastructure presented in Section 4.2 deals with most of the requirements on the metadata as presented in Section 2.1.2. Some of these requirements are met through the expressivity offered by the data security taxonomy proposed, for example, the requirements of data integrity, root metadata traceability, data fine-grained access and usage control, data usage reporting and quality management and conflict classes.

The requirements of metadata interoperability, derivation and offline operability will have to be incorporated as features or functionalities of the MMS system.

5 Bibliography

Ankolekar, A., Burstein, M. H., Hobbs, J. R., Lassila, O., Martin, D. L., McIlraith, S. A., et al. (2001). DAML-S: Semantic Markup for Web Services. *Proceedings of First Semantic Web Working Symposium*, (pp. 411-430).

Ashley, P., Hada, S., Karjoth, G., Powers, C., & Schunter, M. (2003). The Enterprise Privacy Authorization Language (EPAL) v1.2. (C. Powers, & M. Schunter, Eds.) IBM.

Authentica. (2005). *Enterprise rights management for document protection.* White paper.

Baader, F., Calvanese, D., McGuinness, D. L. , Nardi, D., Patel-Schneider, P. F. (2003). *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press.

Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E. (2008). *XML Signature Syntax and Processing (Second Edition)*, W3C Recommendation.

BBSRC. (n.d.). *BBSRC's data sharing policy.* Retrieved 10 1, 2008 from Biotechnology and Biological Sciences Research Council: http://www.bbsrc.ac.uk/publications/policy/data_sharing_policy.pdf

Becker, M. Y., Fournet, C., & Gordon, A. D. (2005). *SecPAL: Design and semantics of a decentralized authorization language.* Microsoft.

Bechhofer, S., Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A. (2004). *OWL Web Ontology Language*. W3C Recommendation.

Lee, T.B., Connolly, D., Kagal, L., Scharf, Y., Hendler, J. (2008). N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, 8(3), 249-269, Cambridge Press".

Bertino, E., Braun, M., Castano, S., Ferrari, E., & Mesiti, M. (2000). Author-X: A java based system for xml data protection. *IFIP TC11/ WG11.3 14th An. Conf. on atabase Security (DBSec)*, (pp. 15-26). Schoorl, The Netherlands.

Blaze, M., Feigenbaum, J., & Keromytis, A. D. (1999). *The KeyNote Trust Management System Version 2.* At & T Labs Research and Univ of Pennsylvania.

Blaze, M., Feigenbaum, J., & Lacy, J. (1996). Decentralized Trust Management. *IEEE Symposium on Security and Privacy* (pp. 164-173). IEEE Computer Society Press.

Boddy, M. (2004). *Data policy and data archiving: Report on consultation.* A report to ESRC.

Boneh, D., & Franklin, M. (2001). Identity-based Encryption from the Weil Pairing. *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology* (pp. 213-229). London, UK: Springer-Verilag.

CCLRC. (2002, July). *Information Systems Privacy and Security Policy.* Retrieved October 1, 2008 from The Council of the Central Laboratory of the Research Councils: http://dl.stfc.ac.uk/DL/Documents/PDF/issue2_0.pdf

Content Guard. (2001). Exensible Rights Markup Language (XrML) 2.0.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. (2001). *Web Services Description Language (WSDL) 1.1.* W3C Note.

Chu, Y.-H., Feigenbaum, J., LaMacchia, B. A., Resnick, P., & Strauss, M. (1997). REFEREE: Trust Management for Web Applications. *Computer Networks , 29* (8-13), 953-964.

Clarke, D., Elien, J.-e., Ellison, C., Fredette, M., Morcos, A., & Rivest, R. L. (2001). Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security , 9* (4), 285-322.

Connolly, D., Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A. (2001). *DAML+OIL (March 2001) Reference Description*. W3C Note.

DCA. (2003, November). *Public Sector Data Sharing: Guidance on the law.* Retrieved 10 1, 2008 from http://www.creatingexcellence.org.uk/uploads/seld/DCA_Public_Sector_Data_Sharing_Guide.pdf

Denker, G., Kagal, L., Finin, T.W., Paolucci, M., Sycara, K.P. (2003). Security for DAML Web Services: Annotation and Matchmaking. *Second International Semantic Web Conference* (335-350), Sanibel Island, FL, USA, Lecture Notes in Computer Science 2870 Springer.

Fikes, R., Jenkins, J., Frank, G. (2003). JTP: A System Architecture and Component Library for Hybrid Reasoning. *Seventh World Multiconference on Systemics, Cybernetics, and Informatics*, Orlando, USA.

P. Hayes (2004). *RDF Semantics*. W3C Recommendation.

Herzberg, A., Mass, Y., Mihaeli, J., Naor, D., & Ravid, Y. (2000). Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. *IEEE Symposium on Security and Privacy*, (pp. 2-14). Oakland.

Hilty, M., Pretschner, A., Basin, D. A., Schaefer, C., & Walter, T. (2008). A Policy Language for Distributed Usage Control. *ESORICS 2008: 13th European Symposium on Research in Computer Security* (pp. 531-546). Springer-Verilag.

Horrocks, I. Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, W3C Member Submission.

Iannella, R. (2002). *Open Digiral Rights Language (ODRL), Version 1.1.* W3C Note, World Wide Web Consortium.

Jajodia, S., Samarati, P., Sapino, M. L., & Subrahmanian, V. S. (2001). Flexible Support for Multiple Access Control Policies. *ACM Trans. on Database Systems , 26* (2), 214-260.

Kagal, L. (2002). *Rei: A Policy Language for the Me-centric Project.* Tech. report, HP Laboratories, Enterprise Systems Data Management Laboratory, Palo Alto.

Kagal, L., Paoucci, M., Srinivasan, N., Denker, G., Finin, T., Sycara, K. (2004). Authorization and Privacy for Semantic Web Services. *IEEE Intelligent Systems (Special Issue on Semantic Web Services)*, 19(4), 50-56, IEEE.

Kagal, K., Lee, T. B., Connolly, D., Weitzner, D. J. (2006). Using Semantic Web Technologies for Policy Management on the Web. *Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*. Boston, Massachusetts, USA, AAAI Press.

Kim, A., Luo, J., Kang, M.H. (2005). Security Ontology for Annotating Resources. *OTM Conferences (2)* (1483-1499).

Kudo, M., & Hada, S. (2000). XML document security based on provisional authorization. *CCS '00: Proceedings of the 7th ACM conference on Computer and communication security* (pp. 87-96). New York: ACM.

Lassila, O., & Swick, R. (1999). *Resource Description Framework (RDF) Model and Syntax*.

W3C Recommendation.

Lee, J. K., & Sohn, M. M. (2003). The eXtensible Rule Markup Language. *Communications of the ACM*, 46(5), 59-64, ACM Press.

Lee, A. J., & Winslett, M. (2006). Safety and consistency in policy-based authorization systems. *Proceedings of the 13th ACM conference on Computer and communications security* (pp. 124-133). Alexandria, Virginia, USAS: ACM.

Li, N., Mitchell, J. C., & Winsborough, W. H. (2002). Design of a Role-based Trust Management Framework. *IEEE Symposium on Security and Privacy*, (pp. 114-130).

Lobo, J., Bhatia, R., & Naqvi, S. (1999). A Policy Description Language. *AAAI'99: Proceedings of the sixteenth national conference on Artificial intelligence* (pp. 291-298). Menlo Park, CA: American Association for Artificial Intelligence.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K. (2004). *OWL-S: Semantic Markup for Web Services.*W3C Member Submission.

Medical Research Council. (2003). *MRC Ethic Series: Personal Information in Medical Research.* London: Medical Research Council.

Microsoft Corp. (2005). *Technical overview of windows rights management services for windows server 2003.*

Mont, M. C., Pearson, S., & Bramhall, P. (2003). Towards Accountable Management of Identity and Privacy: Sticky Policies and Enforceable Tracing Services. *14th International Workshop on Database and Expert Systems Applications* (pp. 377-382). Washington DC, USA: IEEE Computer Society.

Moses, T. (2005). eXtensible Access Control Markup Language (XACML) version 2.0. Oasis Open.

Park, J., & Sandhu, R. (2004). The UCONabc usage control model. *Transactions in Information System Security , 7* (1), 128-174.

Pnueli, A. (1979). The Temporal Semantics of Concurrent Programs. *Proceedings of the International Symposium of Concurrent Computation* (pp. 1-20). London: Springer-Verilag.

Prevelakis, V., Morin, J.-H., & Konstantas, D. (1999). Controlling the dissemination of electronic documents. *10th Int. Workshop on Database and Expert Systems Applications (DEXA)*, (pp. 869-873). Florence, Italy.

Sandhu, R. S., Zhang, X., Ranganathan, K., & Covington, M. J. (2006). Client-side access control enforcement using trusted computing and pei models. *J. High Speed Networks , 15* (3), 229-245.

Sufi, S., Matthews, B., Kleese van Dam, K. (2003). An Interdisciplinary Model for the Representation of Scientific Studies and Associated Data Holdings. *Proceedings of the All Hands Meeting (AHM)*, Nottingham, UK, September 2003.

Sufi, S., Matthews, B. (2004). CCLRC Scientific Metadata Model: Version 2. *Technical Report DL-TR-2004-001*, CCLRC, 2004.

TCPA. (2001). *Main Specification v1.1.* From Trusted Computing Platform Alliance: www.trustedcomputing.org

Twidle, K., Lupu, E., Dulay, N., & Sloman, M. (2008). Ponder2 - A Policy Environment for Autonomous Prevasive Systems. *POLICY 2008: Proceedings of Workshop on Policies for Distributed Systems and Networks* (pp. 245-246). New York: IEEE Computer Society.

Uszok, A., Bradshaw, J. M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., Aitken, J. A. (2004). KAoS Policy Management for Semantic Web Services. *IEEE Intelligent Systems*, 19(4), 32-41.

Wright, G. H. (1951). Deontic Logic. *Mind*, 60, (1-15).

Appendix 1. Policy Language Syntax

```
relationalOp ::= > | < | = | <= | >=;

booleanOp ::= AND | OR;

combineOp ::= AND | OR;

RemoteServiceDeclaration ::= RemoteService serviceName = serviceID;

CertificationAuthorityDeclaration ::= CertAuhtority authorityName =
auhtorityID;

CredentialDeclaration ::= Credential credentialName = (Attribute_list);


Attribute_list ::= Attribute_list, Attribute_list | AttributeID;


complex_condition ::= (complex_condition booleanOp complex_condition) |
atomic_condition;


atomic_condition ::= (subjectName.attributeName | targetName.attributeName
| action-eventParameter | [serviceName.]predicateName
(actual_parameter_list) relationalOp value;
```

```
complex_action_list::= (complex_action_list combineOp complex_action_list)
| temporalOp (tempCond, complex_action_list) |
atomic_action;

atomic_action::= [entityName.]actionName(actual_parameter_list);
```

Authentication policies

```
RoleDeclaration ::= role roleName requires credential_list;

credential_list ::= (credential_list booleanOp credential_list) |
atomic_credential |
null;

atomic_credential ::= integer credentialName(complex_condition) signed by
[variableName:]entityName;
```

Authorisation policies

```
AuthorisationPolicy ::=
Authorisation PolicyName = allow | deny actionName (parameter_list) filter
(filter_specification)
     target [variableName:]ObjectExpression
    [to [variableName:]entityName]
         [with (credential_list)]
         [when (complex_condition)]
         [while (complex_condition)]
         [preObligation (complex_action_list)]
         [postObligation (complex_action_list)];
```

Event-Condition-Action policies

```
ECAPolicy ::=
on eventName (parameter_list)
    do complex_action_list
         [when (complex_condition)];
```